Heuristic Optimization of Boolean Functions and Substitution Boxes for Cryptography

by

Linda Burnett

Previous qualifications – Bachelor of Applied Science (Honours) 1997

Thesis submitted in accordance with the regulations for Degree of Doctor of Philosophy

Information Security Institute
Faculty of Information Technology
Queensland University of Technology

2005
Keywords

boolean function, substitution box, heuristic techniques, Genetic Algorithm, Hill Climbing, cryptographic properties, autocorrelation, balance, nonlinearity, correlation immunity, resilience, propagation criteria
Abstract

Fundamental to the electronic security of information and communication systems, is the correct use and application of appropriate ciphers. The strength of these ciphers, particularly in their ability to resist cryptanalytic attacks, directly influences the overall strength of the entire system. The strength of the underlying cipher is reliant upon a robust structure and the carefully designed interaction between components in its architecture. Most importantly, however, cipher strength is critically dependent on the strength of the individual components of which it is comprised.

Boolean functions and substitution boxes (s-boxes) are among the most common and essential components of ciphers. This is because they are able to provide a cipher with strengthening properties to resist known and potential cryptanalytic attacks. Thus, it is not surprising that significant research effort has been made in trying to develop ways of obtaining boolean functions and substitution boxes with optimal achievable measures of desirable cryptographic properties. Three of the main cryptographic properties required by strong boolean functions and s-boxes are nonlinearity, correlation immunity and propagation criteria, with different cryptographic applications requiring different acceptable measures of these and other properties. As combinations of cryptographic properties exhibited by functions can be conflicting, finding cryptographically strong functions often means that a trade-off needs to be made when optimizing property values. Throughout this thesis, the term “optimization” specifically refers to seeking to obtain the best achievable combination of target property values which may be exhibited by boolean functions and s-boxes, regardless of whether the relevant properties are conflicting or complementary.

This thesis focusses on a particular class of techniques for obtaining strong functions for cryptographic applications, referred to as heuristic methods or, simply, heuristics. Three new heuristic methods, each aimed at generating boolean functions optimizing one or more of the main cryptographic properties mentioned
above, in addition to other desirable properties, are presented. The first of the new heuristic methods developed for this thesis focusses on generating boolean functions which are balanced and exhibit very high nonlinearities. Highly non-linear balanced functions are critical to many cryptographic applications, as they provide good resistance to linear cryptanalytic attacks. This first method is based on the recursive modification of a starting bent function and is shown to be highly successful and efficient at generating numerous such functions, which also exhibit low autocorrelation values, in a very short computational time.

The generation of balanced, correlation immune boolean functions that also exhibit the conflicting property of high nonlinearity is the focus of the second new heuristic method developed for this thesis. By concatenating selected pairs of lower-dimensional boolean functions together in the Walsh Hadamard transform domain, direct optimization for both resilience and nonlinearity was able to take place at each level towards and for the final function. This second method was able to generate examples of boolean functions with almost all of the best known optimal combinations of target property values. Experiments have shown the success of this method in consistently generating highly nonlinear resilient boolean functions, for a range of orders of resilience, with such functions possessing optimal algebraic degree.

A third new heuristic method, which searches for balanced boolean functions which satisfy a non-zero degree of propagation criteria and exhibit high nonlinearity, is presented. Intelligent bit manipulations in the truth table of starting functions, based on fundamental relationships between boolean function transforms and measures, provide the design rationale for this method. Two new function generation schemes have been proposed for this method, to efficiently satisfy the requirements placed on the starting functions utilized in the computational process. An optional process attempts to increase the algebraic degree of the resulting functions, without sacrificing the optimalities that are achievable. The validity of this method is demonstrated through the success of various experimental trials.

Switching the focus from single output boolean functions to multiple output boolean functions (s-boxes), the effectiveness of existing heuristic techniques (namely Genetic Algorithm, Hill Climbing Method and combined Genetic Algorithm/Hill Climbing) in primarily being applied to improve the nonlinearity of s-boxes of various dimensions, is investigated. The prior success of these heuristic
techniques for improving the nonlinearity of boolean functions has been previously demonstrated, as has the success of hill climbing in isolation when applied to bijective s-boxes. An extension to the bijective s-box optimization work is presented in this thesis. In this new research, a Genetic Algorithm, Hill Climbing Method and the two in combination are applied to the nonlinearity and autocorrelation optimization of regular $N \times M$ s-boxes ($N > M$) to investigate the effectiveness and efficiency of each of these heuristics. A new breeding scheme, utilized in the Genetic Algorithm and combined Genetic Algorithm/Hill Climbing trials, is also presented. The success of experimental results compared to random regular s-box generation is demonstrated.

New research in applying the Hill Climbing Method to construct $N \times M$ s-boxes ($N < M$) required to meet specific property criteria is presented. The consideration of the characteristics desired by the constructed s-boxes largely dictated the generation process. A discussion on the generation process of the component functions is included. Part of the results produced by experimental trials were incorporated into a commonly used family of stream ciphers, thus further supporting the use of heuristic techniques as a useful means of obtaining strong functions suitable for incorporation into practical ciphers.

An analysis of the cryptographic properties of the s-box used in the MARS block cipher, the method of generation and the computational time taken to obtain this s-box, led to the new research reported in this thesis on the generation of MARS-like s-boxes. It is shown that the application of the Hill Climbing Method, with suitable requirements placed on the component boolean functions, was able to generate multiple MARS-like s-boxes which satisfied the MARS s-box requirements and provided additional properties. This new work represented an alternative approach to the generation of s-boxes satisfying the MARS s-box property requirements but which are cryptographically superior and can be obtained in a fraction of the time than that which was taken to produce the MARS s-box. An example MARS-like s-box is presented in this thesis.

The overall value of heuristic methods in generating strong boolean functions and substitution boxes is clearly demonstrated in this thesis. This thesis has made several significant contributions to the field, both in the development of new, specialized heuristic methods capable of generating strong boolean functions, and in the analysis and optimization of substitution boxes, the latter achieved through applying existing heuristic techniques.
Contents

Keywords iii

Abstract v

Declaration xvii

Previously Published Material xix

Acknowledgements xxi

1 Introduction 1
  1.1 Objectives and Outcomes .......................... 3
  1.2 Structure of Thesis .................................. 6

2 Review of Boolean Function and S-Box Theory 11
  2.1 Boolean Function Theory ............................... 11
     2.1.1 Characteristics of Boolean Functions .............. 12
     2.1.2 Cryptographic Properties of Boolean Functions .... 18
         Balance ............................................... 18
         Nonlinearity ....................................... 19
         Avalanche ......................................... 24
         Correlation Immunity ............................... 30
     2.1.3 Relationships Between Cryptographic Properties of Boolean Functions ................. 32
         Nonlinearity and Avalanche ....................... 32
         Nonlinearity and Correlation Immunity ............. 35
         Correlation Immunity and Avalanche ............... 36
     2.1.4 Some Special Boolean Functions ................. 37
         Bent Functions .................................... 37
# Table of Contents

38 Semi-Bent Functions  
39 Plateaued Functions  
40 2.2 S-Box Theory  
41 2.2.1 S-Box Definitions and Types  
42 2.2.2 Cryptographic Properties of S-Boxes  
44 2.3 Some Common Cryptanalytic Attacks on Cipher Systems  
45 2.3.1 Differential Cryptanalysis  
47 2.3.2 Linear Cryptanalysis  
50 2.3.3 Correlation Attacks  
51 2.4 Summary  
53 3 Heuristic Techniques  
54 3.1 Overview of Existing Heuristic Techniques Used  
55 3.1.1 Hill Climbing  
56 Experimental Rationale  
58 Previously Reported Results  
59 Method Applicability  
59 3.1.2 Genetic Algorithms  
62 Experimental Rationale  
63 Previously Reported Results  
64 Method Applicability  
64 3.1.3 Combined Genetic Algorithm and Hill Climbing  
67 3.2 Summary  
69 4 The Development and Application of New Heuristic Methods to Boolean Function Property Optimization  
71 4.1 Highly Nonlinear Cryptographically Strong Boolean Functions  
71 4.1.1 Related Work by Other Researchers  
73 4.1.2 Method 1  
77 Experimental Rationale  
78 Experimental Results  
85 Method Applicability  
85 4.2 Resilient Boolean Functions  
95 Experimental Rationale  
96 Method Applicability  
98 Experimental Results  
103 Method Applicability  
104 5 Summary  
104 5.1 Experimental Rationale  
106 Method Applicability  
109 Experimental Results  
112 Method Applicability  
114 6 Conclusions  
114 6.1 Research Work Summary  
117 6.2 Future Work  
119 References  
121 6.3 Acknowledgements  
127 6.4 Bibliography  
130
4.2.1 Related Work by Other Researchers ......................... 86
4.2.2 Method 2 ...................................................... 88
    Experimental Rationale ........................................ 91
    Experimental Results .......................................... 91
    Method Applicability .......................................... 98

4.3 Boolean Functions Satisfying \( PC(k) \) of Order 0 ......................... 100
4.3.1 Related Work by Other Researchers ......................... 100
4.3.2 Method 3 ...................................................... 101
    Experimental Rationale ........................................ 107
    Experimental Results .......................................... 108
    Method Applicability .......................................... 117

4.4 Summary ........................................................ 119

5 Application of Heuristic Techniques to Substitution Box Analysis
    and Property Optimization ......................................... 123
5.1 \( N \times M \) Regular S-Box Generation \((N > M)\) using Genetic Algo-
    rithm/Hill Climbing ............................................. 124
    5.1.1 Experimental Rationale ..................................... 125
    5.1.2 Experimental Results ...................................... 128
    5.1.3 Method Applicability ...................................... 138
5.2 An Example of Practical \( N \times M \) S-Box Generation \((N < M)\) ... 139
    5.2.1 Desired Characteristics of Qualcomm S-Boxes ............ 140
    5.2.2 Techniques Used for Generation of 8x16 and 8x32 S-Boxes 141
    5.2.3 Experimental Results ...................................... 141
    5.2.4 Practical Use of S-Boxes .................................. 144
5.3 Summary ........................................................ 145

6 Practical Application of Heuristic Techniques to MARS-Like S-
    Box Generation .................................................... 147
6.1 The Block Cipher, MARS .......................................... 148
    6.1.1 General Design of MARS .................................. 149
    6.1.2 Usage of S-Boxes in MARS ................................ 150
    6.1.3 MARS S-Box Property Requirements ...................... 151
    6.1.4 Technique used for MARS S-Box Generation .............. 157
6.2 MARS-like S-Box Generation .................................... 158
List of Figures

3.1 Algorithm: Hill Climbing Method ........................................ 57
3.2 Algorithm: Genetic Algorithm ............................................. 61
3.3 Algorithm: Combined Genetic Algorithm/Hill Climbing ............. 65
4.1 Algorithm: Method 1 ....................................................... 76
4.2 Example of a 6-variable boolean function (in hex notation) with
   $\text{NL} = 26, \text{deg} = 5, \text{AC}_{\text{max}} = 16$ and $\sigma = 6,784$ ........ 80
4.3 Example of an 8-variable boolean function (in hex notation) with
   $\text{NL} = 116, \text{deg} = 7, \text{AC}_{\text{max}} = 16$ and $\sigma = 89,728$ ........ 81
4.4 Example of a 10-variable boolean function (in hex notation) with
   $\text{NL} = 488, \text{deg} = 9, \text{AC}_{\text{max}} = 40$ and $\sigma = 1,272,448$ ....... 81
4.5 Example of a 12-variable boolean function (in hex notation) with
   $\text{NL} = 2000, \text{deg} = 11, \text{AC}_{\text{max}} = 64$ and $\sigma = 18,757,120$ .... 81
4.6 Example of a 6-variable balanced boolean function with
   $\text{NL} = 26, \text{deg} = 5, \text{AC}_{\text{max}} = 16, \sigma = 6784$ and satisfying $SAC$ ....... 83
4.7 Algorithm: Method 2 ....................................................... 89
4.8 Example of a (7,2,4,56) balanced boolean function .................. 93
4.9 Example of an (8,1,6,112) balanced boolean function ............... 93
4.10 Example of a (9,2,5,240) balanced boolean function ............... 94
4.11 Algorithm: Method 3 ..................................................... 103
4.12 Algorithm: Starting Function Generation Scheme A ................. 105
4.13 Algorithm: Starting Function Generation Scheme B ................. 106
4.14 Example of a (6,2,5,26,16,6784) balanced boolean function ...... 111
4.15 Example of a (6,1,4,24,24,10240) balanced boolean function ...... 111
4.16 Example of a (7,2,5,56,48,26624) balanced boolean function ...... 112
4.17 Example of a (7,1,5,52,32,35840) balanced boolean function ...... 112
4.18 Example of a (7,1,3,56,64,32768) balanced boolean function ...... 113
4.19 Example of an (8,1,6,116,24,93952) balanced boolean function ... 113
4.20 Example of an \((8,1,6,112,48,152320)\) balanced boolean function

4.21 Example of a \((6,1,4,24,24,10240)\) balanced \(CI(1)\) boolean function

4.22 Example of a \((7,1,3,56,64,32768)\) balanced \(CI(1)\) boolean function

5.1 \(breed(S1, S2)\)

5.2 Nonlinearity -v- Frequency, comparing Genetic Algorithm with Random Regular 8x4 s-box generation

5.3 Autocorrelation -v- Frequency, comparing Genetic Algorithm with Random Regular 8x4 s-box generation

5.4 Nonlinearity -v- Frequency, comparing Hill Climbing with Random Regular 8x4 s-box generation

5.5 Nonlinearity -v- Frequency, comparing Genetic Algorithm with Combined Genetic Algorithm/Hill Climbing and Random Regular 8x4 s-box generation

5.6 Autocorrelation -v- Frequency, comparing Genetic Algorithm with Combined Genetic Algorithm/Hill Climbing and Random Regular 8x4 s-box generation

5.7 Nonlinearity -v- Frequency, comparing Hill Climbing with Genetic Algorithm and Combined Genetic Algorithm/Hill Climbing and Random Regular 8x4 s-box generation

5.8 Best Nonlinearity, comparing Genetic Algorithm with Random Regular s-box generation for \(N = 8\), varying \(M\)

5.9 Best Autocorrelation, comparing Genetic Algorithm with Random Regular s-box generation for \(N = 8\), varying \(M\)

5.10 Genetic Algorithm: Change in Nonlinearity -v- Hamming Distance with Increase in Iterations

5.11 Number of Iterations of Genetic Algorithm until Stopping Criteria Satisfied

5.12 Generation Process for 8x16 and 8x32 s-boxes

6.1 High-Level Structure of MARS from [39]

6.2 Generation Process for MARS-like S-Boxes
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Size of boolean function output space as $N$ increases $1 \leq N \leq 8$</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>Highest nonlinearity found by various heuristic methods for balanced $N$-variable boolean functions, $6 \leq N \leq 14$, $N$ even, compared with Dobbertin’s conjectured maximal nonlinearity values</td>
<td>79</td>
</tr>
<tr>
<td>4.2</td>
<td>Best values for combined nonlinearity ($NL$) and sum-of-square indicator ($\sigma$) measures exhibited by even-dimensional balanced boolean functions generated by Method 1</td>
<td>80</td>
</tr>
<tr>
<td>4.3</td>
<td>Best measures of property value combinations exhibited by even-dimensional balanced boolean functions generated by Method 1</td>
<td>82</td>
</tr>
<tr>
<td>4.4</td>
<td>Order of magnitude of program run times for Method 1 to achieve $N$-variable boolean functions with $6 \leq N \leq 14$, $N$ even</td>
<td>84</td>
</tr>
<tr>
<td>4.5</td>
<td>Optimal combinations of property values known at the time of this research, which were able to be achieved by Method 2. Note, however, that (8,1,6,112) is not optimal for nonlinearity</td>
<td>92</td>
</tr>
<tr>
<td>4.6</td>
<td>Best nonlinearity achieved by Genetic Algorithm in [67] for 1-resilient boolean functions. A dash in the table indicates that no algebraic degree was reported</td>
<td>94</td>
</tr>
<tr>
<td>4.7</td>
<td>Best nonlinearity achieved by Directed Search Algorithm in [78] for 1-resilient boolean functions and also obtaining highest algebraic degree</td>
<td>95</td>
</tr>
<tr>
<td>4.8</td>
<td>Best $(N, m, NL)$ property value combinations $(8 \leq N \leq 12)$ constructed by Algorithm A and Algorithm B in [85]</td>
<td>96</td>
</tr>
<tr>
<td>4.9</td>
<td>Comparison of 8 and 9-variable results, achieved by various methods, focussing on resilience</td>
<td>96</td>
</tr>
<tr>
<td>4.10</td>
<td>Combination of property values achieved by [17] using simulated annealing where $5 \leq N \leq 10$</td>
<td>98</td>
</tr>
</tbody>
</table>
4.11 Best measures of property value combinations exhibited by boolean functions generated by Method 3 for $N \in \{4,5,..,10\}$. Note that an * indicates boolean functions which are bent and therefore unbalanced. The absence of an * indicates that balanced boolean functions with these combined property values were found.

4.12 Typical parameter values used to achieve the best functions using Method 3.

4.13 Best nonlinearity and $AC_{\text{max}}$ values achieved by $PC(1)CI(1)$ balanced boolean functions reported in [17].

4.14 Comparison of 6, 7 and 8-variable results achieved by [17] and Method 3. An * indicates that the functions are able to be transformed into $CI(1)$ functions.

4.15 Order of magnitude of program run times for Method 3 to achieve $PC(1)$ $N$-variable boolean functions with $4 \leq N \leq 10$ and targeting the given nonlinearity values.

5.1 Value and frequency of best nonlinearity achieved by the Genetic Algorithm, and rate of improvement with final pool hill climbed.

5.2 Best s-boxes in terms of values exhibited by component boolean functions represented as <balanced, minimum nonlinearity, maximum deviation from $CI(1)$ for first 8, maximum imbalance between pairs>.

6.1 MARS s-box: Satisfaction of Differential and Linear Property Requirements.

6.2 MARS-like s-box: Satisfaction of MARS Differential and Linear Property Requirements.

6.3 MARS and MARS-like s-boxes: Satisfaction of Differential and Linear Property Requirements.
Declaration

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed: ................................... Date: ..........................
Previously Published Material

The following papers have been published or presented, and contain material based on the content of this thesis.


Acknowledgements

Typically a PhD student has many people to thank during the course of their PhD and for the completion of the PhD being reached. I can honestly say that only a small group of people are responsible for this particular PhD achievement being possible, but the quantity of the thanks I wish to give to these people is no small measure of my gratitude. In no particular order I give my specific thanks to these people.

Thank you to Gary Carter. The joint work we did as students together in the early days was productive. Our work and discussions helped us to gain knowledge together.

Thanks to my principal supervisor, Ed Dawson, for his encouragement. His demands, however unrealistic at times, are the greatest proof that he sincerely wants his students to succeed.

I wish to thank Andrew Clark. I am mostly grateful to him for his time. It is not uncommon as a student to feel that you are lowest in the hierarchical structure. Although extremely busy most of the time, Andrew always made himself available to listen to my queries and give me advice on research matters, particularly of a programming nature. I hope that I have always been considerate of his time. He has been an objective and supportive supervisor.

Thank you to Bill Millan for the opportunity to learn from his vast knowledge and experience in this field and for sharing his countless ideas for new work, and contagious enthusiasm for this area.

Thank you to Qualcomm Australia for providing their kind permission to report on the research work performed for their SOBER family of stream ciphers.

My deepest thanks goes to Mark Looi for his unwavering support. I am forever grateful to him for his strength and encouragement.
Chapter 1

Introduction

Today, civilization is firmly within the realm of the Information Age, which is characterized by intellectual property and useable inside knowledge being considered extremely valuable. Information exists and is used in many forms including financial (transactions), legal (documents), military (plans and strategies), and political. The protection of this information, both in storage, transit and in day-to-day usage, is vital as its compromise may result in financial loss, exposure of commercial or military secrets and even the loss of life.

Cryptography is one of the most important mechanisms used in the provision of information security. Three main forms of protection are offered by cryptography through the use of suitable and well structured cryptographic cipher systems. These are confidentiality, integrity and authentication. Confidentiality is provided by ensuring that information is kept private from unauthorized disclosure. Integrity is provided by making sure the information has not been modified since creation or storage either maliciously or unintentionally. Authentication is the process of checking that the sender of the information is correctly identified and legitimate.

Cryptographic algorithms are often categorized by factors such as their key distribution principles and the size of their input stream. Symmetric ciphers have a common secret key shared between communicating parties, whilst asymmetric ciphers use different (unrelated) keys for encryption and decryption. Symmetric ciphers can be further categorized into block or stream ciphers where the input to the cipher takes the form of either blocks of data or continuous bitstreams re-
spectively. Another type of cryptographic cipher system which compresses data to form a digest in order to provide integrity and/or authentication is a cryptographic hash function.

Cipher systems are a prime target for an attacker wishing to compromise the information being protected by a security system. In line with the three forms of protection mentioned above, the typical motives of an attacker include seeking to reveal confidential information, to illicitly and surreptitiously modify information, and to falsely claim an identity. In addition, an attacker may seek to remove evidence, or even insert false evidence, that a particular event or transaction has occurred. Compromising a cipher system which endeavours to protect this information can either directly enable these actions to occur, or indirectly weaken another part of the system to enable these actions to later occur. Powerful existing cryptanalytic attacks against cipher systems have proved to be successful under the right conditions.

The overall strength of a security system is dependent on the strength of the individual components, such as the authentication system, the key management system, the data storage system, the cipher system and the policies and procedures, to name a few. Similarly, the overall strength of a cipher system is dependent on the strength of its individual components. A weakness in any of the individual components may lead to a catastrophic failure in the whole cipher.

Boolean functions and substitution boxes (s-boxes) are two of the most common and critical components of cryptographic cipher systems. These components are directly related by function quantity. That is, a substitution box is typically comprised of multiple single output boolean functions, but if it maps to only one bit, is identical to a boolean function.

Boolean functions are often utilized in the keystream generation process of stream ciphers as they are highly suitable for receiving bits of linear feedback shift registers as input in order to combine them as securely as possible to produce the single keystream. Further, boolean functions are capable of exhibiting the combination of cryptographic properties necessary to resist the typical types of attacks which seek to reveal part or all of the keystream.

It is common for cryptographic hash functions to incorporate boolean functions in their round functions for the strengthening properties (particularly in terms of diffusion) they are able to contribute to the compression process. Their computational efficiency is also highly sought as fast hashing is an important
design consideration.

The most common type of cipher system which employs s-boxes are block ciphers. As a block cipher system encrypts its data in fixed length blocks, s-boxes are a natural component of such a system. They provide a means of substituting multiple bits (part of or a whole block) of data for a completely different set of output bits. More importantly is the use of strong s-boxes (those which possess good cryptographic properties) so the substitution signifies a complex relationship between input and output bits of the s-box. The typical use of s-boxes in the cipher's iterative round function serves to increase the effort needed to exploit any statistical structure in the data.

Boolean functions and s-boxes will only be able to contribute to the security of a cipher by possessing good measures of desirable cryptographic properties. Obtaining strong boolean functions and s-boxes for incorporation into cryptographic cipher systems to enhance their security is an ongoing research problem. This is particularly so as cryptanalytic techniques become more sophisticated, and with the advancement of computing technology which works both for and against cryptographic security. The size of boolean functions and the dimension of s-boxes have a significant bearing on security. Though larger functions generally require more computational effort in order to exploit weaknesses, so too is the computational effort increased when attempting to obtain large functions with exceptionally good measures of desirable cryptographic properties. This adds an extra element of difficulty to the research problem.

1.1 Objectives and Outcomes

Listed below are the main objectives of the research presented in this thesis:

1. to contribute to the ongoing problem of obtaining strong boolean functions and s-boxes to increase the security of cryptographic cipher systems which utilize them.

These strong cipher components will be those boolean functions and s-boxes which exhibit suitable measures of a combination of cryptographic properties appropriate for their use according to the type of cipher employing them. The task of obtaining such functions involves generating and/or constructing boolean functions and s-boxes which not only exhibit the required measure of properties but are also of a large enough dimension that they
are able to provide resistance to attacks in the long term. Additionally, the means of obtaining these strong cipher components must be reasonably computationally efficient.

2. to apply known heuristic techniques to optimize target s-box properties and to determine their effectiveness in doing so.

The aim will be to conduct extensive experimentation using known heuristic techniques to establish their effectiveness and suitability in optimizing specific s-box properties for s-boxes of practical dimension. The computational feasibility of using these methods will be determined and comparisons made between each method in terms of achievable property measures and computational efficiency.

3. to use appropriate results from the experiments in 2. for incorporation into existing cryptographic ciphers and to aid in the analysis of existing cipher components.

The aim is to use the observations and knowledge gained from the experimental s-box trials to improve on existing cipher components and to incorporate strong s-boxes which we have generated into existing ciphers.

4. to develop new heuristic methods suitable for obtaining strong boolean functions and s-boxes.

A sensible approach to the problem of obtaining strong boolean functions and s-boxes which may be suitable as cipher components is to design and develop new methods with which they can be generated or constructed. To this end, we aim to develop some new heuristic methods in this thesis for the application to this task, in addition to trialling known heuristic techniques.

5. to determine the effectiveness of these new heuristic methods applied to boolean function property optimization.

Extensive experimental trials need to be conducted to ascertain the degree of success of the new heuristic methods developed for this thesis in generating boolean functions exhibiting good measures in target properties.

The outcomes of the work in this thesis are now discussed with reference to the above objectives.
In this thesis the approach that we have taken is to apply heuristic techniques to the generation of boolean functions and s-boxes in order to improve target cryptographic properties. This new research work, which has involved significant programming effort and extensive experimental trials, has enabled the determination of how effective and efficient the trialled heuristics are at this task. A secondary outcome of this work was the direct observation and understanding of the nature of cryptographic properties and the extent to which properties can exist in combination. Specifically, to address objective 1, we have applied known heuristic techniques to s-box property optimization and developed three new heuristic methods for boolean function property optimization, and achieved excellent results from our experiments.

With respect to objective 2, we have conducted numerous experimental trials on the application of two existing heuristic techniques individually and in combination to improve upon target s-box properties. This allowed us to ascertain their effectiveness and to make comparisons between these techniques and their variations in relation to achievable property values and computational efficiency. Thus, it has been possible to determine the suitability of each for this task. The research work performed for this thesis relating to this outcome was published in [A3].

Objective 3 was satisfied in large part by careful analysis of property requirements, together with additional experimentation which was cipher specific. An important outcome of this work was that requirements appropriate for the s-box generation task were able to be determined and the most effective heuristic technique used. Some of the best results achieved by our s-box experiments were incorporated into a number of stream ciphers of the SOBER family. Other s-box work which was conducted involved the analysis of the MARS [39] s-box properties. Through this research we were able to develop an alternative s-box generation process which was significantly more efficient and was also capable of generating stronger s-boxes (MARS-like). This new research is of great practical significance and clearly justifies the valuable contribution of the research in this thesis to the security of cryptographic cipher systems. The MARS-like s-box research has been published as [A1].

The aim of objective 4 was to develop one or more new heuristic methods to be used in order to contribute to the ongoing research problem of obtaining strong cipher components. The outcome of this objective was that three new
heuristic methods have been developed, coded and applied in this thesis to the rapid generation of many high quality boolean functions. These new methods were designed to target specific combinations of cryptographic properties. [A2] is the relevant publication pertaining to two of the new developed methods arising from this outcome.

The manner in which the task involving objective 5 was performed was by conducting numerous experiments, collating results, making observations and analyzing the results to ascertain the effectiveness of the three new heuristic methods. Experimental results have shown our methods to be highly successful. Boolean functions exhibiting many of the best known combinations of cryptographic property values have been achievable by certain of our methods. Further, other heuristic methods have been unable to obtain boolean functions with some of the best combinations of property measures which our methods were able to achieve. The application of two of the new heuristic methods, including experimental results, was included in [A2].

1.2 Structure of Thesis

This thesis is divided into seven chapters, including this introductory chapter. Chapter 2 presents a summary review of the boolean function and s-box theory which is relevant to the work in this thesis. This chapter is a reference point for the necessary background material required to understand the research and its importance in the field of cryptography and information security. The inclusion of numerous long established definitions and theorems relating to boolean functions such as those involving representations and measures of significance, provide the essential structure for recognizing and understanding their characteristics and limitations. This chapter also defines and discusses many important cryptographic properties of boolean functions and their extension to s-boxes. Following on from this is a brief discussion on the relationship between particular pairs of properties and to what extent they may co-exist within the same function. Some special boolean functions which are referred to throughout this thesis are defined herein and details of their unique characteristics are given. This chapter would not be complete without a brief discussion on a few of the most common cryptanalytic attack techniques which are applied to cipher systems and their components, and the role played by specific cryptographic properties in their
1.2. Structure of Thesis

resistance.

Chapter 3 provides an overview of the existing heuristic techniques which have been applied in this thesis for s-box optimization. This includes a description of their algorithms and a discussion on the process involved for each technique. This chapter is essentially an overview of two important known heuristic techniques, Hill Climbing and Genetic Algorithms, and also discusses their combined use. A brief summary of previously reported results of past work using the Hill Climbing Method and a Genetic Algorithm, and the two in combination, is given for completeness.

Chapter 4 represents new research work in this area and contains the descriptions and algorithms of three new heuristic methods that have been developed for this thesis. The first method proposes a novel technique involving small recursive changes to bent boolean functions to optimize the nonlinearity property and achieve low autocorrelation with the aim of generating a large number of strong, balanced boolean functions with considerably good measures for these properties. The second method developed for this thesis is designed to generate highly nonlinear resilient boolean functions by operating directly in the Walsh Hadamard transform domain with careful manipulation of concatenation layers. This exciting new approach allows two conflicting properties to be simultaneously optimized and optimum tradeoffs to be achieved. The third new method proposed in this thesis uses the separate relationships between specific representations, transformations and measures of a boolean function to simultaneously optimize and/or satisfy three cryptographic properties of boolean functions by selective bit manipulation. This method thus enables the generation of balanced boolean functions satisfying propagation criteria of non-zero degree with high nonlinearity.

In addition to the descriptions and algorithms for our three new developed heuristic methods provided in Chapter 4, a discussion on the manner in which they set out to achieve their target properties and the process involved in the computation is also provided. The application and effectiveness of each of these three new methods of function generation and optimization is also detailed and discussed in this chapter.

Further, in Chapter 4 of this thesis, we report on the results of a vast number of experiments which have been conducted by applying each of the new heuristic methods to boolean functions in order to improve their cryptographic properties.
In this chapter we discuss the experimental rationale and justification for each of our new methods, as well as explain the appropriate usage of each method. A number of the best results for each of our new methods is provided and a discussion on the significance of these results given. This chapter also includes examples of some of the best functions we were able to generate together with the parameters used to achieve these results. Comparisons between the results achieved by our developed methods and those attainable by other heuristic techniques further demonstrate the effectiveness and efficiency of our new methods. The results also show that our new heuristic methods each contribute to advancing the overall quality of boolean function generation and optimization in a novel and unique manner.

The new research performed in Chapter 5 of this thesis is concentrated on the optimization of certain cryptographic properties of substitution boxes. In particular, the chapter is divided into two sections each of which considers a contrasting input-output s-box dimensionality type. The first section of Chapter 5 reports on the effectiveness of the Hill Climbing Method and Genetic Algorithm (both described in Chapter 3) to optimize the nonlinearity of \(N \times M\) regular s-boxes \((N > M)\). Extensive experiments using these two methods individually and in combination are reported on 8-variable functions for a range of outputs. A number of consistent results are presented and discussed, from which clear conclusions can be drawn. The new research presented in the second section of this chapter involves the utilization of the Hill Climbing Method applied to randomly generated single output balanced boolean functions in order to construct \(N \times M\) s-boxes with \(N < M\). In particular, this new work aims at constructing 8x16 and 8x32 s-boxes with specific requirements with respect to cryptographic properties and relationships between component functions. Certain results, which were incorporated into a number of stream ciphers of the SOBER family, are of particular practical significance.

Chapter 6 of this thesis contains new research of a practical nature involving an analysis of the 9x32 s-box used in an existing symmetric block cipher, MARS [39], and subsequent application of heuristic techniques in order to generate MARS-like s-boxes with the same or better cryptographic properties. In this chapter we discuss our MARS-like s-box property requirements and report on the technique which we used for MARS-like s-box generation. The results of our generation process in terms of s-box properties and computational effort are compared and
contrasted with those of the MARS s-box. This work provides an alternative and superior technique for MARS-like s-box generation.

The final chapter of this thesis, Chapter 7, presents conclusions about the research performed for this thesis. It also highlights numerous directions for future research in this area.
Chapter 2

Review of Boolean Function and S-Box Theory

The research work reported in this thesis requires knowledge of previously established boolean function and s-box theory. Such knowledge is essential not only for linking the theoretical concepts to practical applications, but also in order to understand the significance of the research and where this work is placed in relation to the field of cryptology. Thus, necessary background is provided in this chapter by stating a number of important long established and well known definitions, theorems and formulae.

In the first section of this chapter, the necessary boolean function theory is provided. This includes discussions on the characteristics and cryptographic properties of boolean functions, as well as the nature of the relationship between various cryptographic properties. The theory of s-boxes is then discussed in the next section, including the definition of various cryptographic properties of s-boxes. Finally, a brief review of some common cryptanalytic attacks is provided.

2.1 Boolean Function Theory

The theory of boolean functions is a broad and extensive area in itself. This section does not purport to be an exhaustive or complete review of boolean function theory. Rather, the theory presented in this section is a comprehensive taxonomy of that which is necessary for the reader to fully understand the research reported
Chapter 2. Review of Boolean Function and S-Box Theory

Table 2.1: Size of boolean function output space as $N$ increases $1 \leq N \leq 8$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$2^N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>65536</td>
</tr>
<tr>
<td>5</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>6</td>
<td>18,446,744,073,709,551,616</td>
</tr>
<tr>
<td>7</td>
<td>340,282,366,920,938,463,374,607,431,768,211,456</td>
</tr>
<tr>
<td>8</td>
<td>115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936</td>
</tr>
</tbody>
</table>

in this thesis. In particular, the cryptographic properties which are relevant to this work are defined and discussed, together with their inter-relationships.

2.1.1 Characteristics of Boolean Functions

We now discuss some important definitions and theorems fundamental to the cryptographic characteristics of boolean functions. This includes specific boolean function measures and a variety of representations used to express a boolean function.

Let $V_2^N = GF(2)^N$ be the $N$-dimensional vector space of binary $N$-tuples. An $N$-variable boolean function $f(x)$ is a mapping from $f(x) : V_2^N \mapsto V_2$ where $x = (x_N, \ldots, x_1)$. Thus, the domain and range of $f(x_N, \ldots, x_1)$ is $\{0, 1\}^N$ and $\{0, 1\}$ respectively.

The total number of distinct $N$-variable boolean functions is $2^{2^N}$. Clearly, as the number of inputs, $N$, increases, the entire boolean function output space grows dramatically. Table 2.1 shows this illustratively for $1 \leq N \leq 8$.

Boolean functions may be represented using a number of forms. Two of the common representations are the truth table and the polarity truth table.

**Definition 2.1.** The truth table of an $N$-variable boolean function, $f(x)$, is the binary output vector of the function and contains $2^N$ elements, each element $\in \{0, 1\}$. 

**Definition 2.2.** The polarity truth table of an $N$-variable boolean function, denoted by $\hat{f}(x)$, is a vector containing $2^N$ elements, each element $\in \{1, -1\}$. It is derived from $f(x)$ by $\hat{f}(x) = 1 - 2f(x)$ or, equivalently, $\hat{f}(x) = -1^f(x)$. 

A useful measure of a boolean function is its hamming weight. Later defini-
tions will show the importance of the hamming weight in terms of cryptographic properties of boolean functions and cryptanalysis.

**Definition 2.3.** Let $hw(f)$ denote the hamming weight of an $N$-variable boolean function, $f(x)$. Then $hw(f) = \sum_{x=0}^{2N-1} f(x)$, the number of ones in the truth table of $f(x)$.

The similarity between two boolean functions can be determined through the use of a measure referred to as the hamming distance.

**Definition 2.4.** Let $hd(f, g)$ denote the hamming distance between two $N$-variable boolean functions, $f(x)$ and $g(x)$. Then $hd(f, g) = \sum_{x=0}^{2N-1} (f(x) \oplus g(x))$, where $\oplus$ denotes addition modulo 2. Thus, $hd(f, g)$ represents the number of differing elements in corresponding positions between the two truth tables.

The correlation between two $N$-variable boolean functions, $f(x)$ and $g(x)$, may be viewed as the degree of similarity between the two functions. This relationship can be expressed in terms of the hamming distance between $f(x)$ and $g(x)$ and is numerically represented as a real number between -1 and 1, known as the correlation coefficient.

**Definition 2.5.** Let $cc(f, g)$ denote the correlation coefficient of two $N$-variable boolean functions, $f(x)$ and $g(x)$. Then

\[
cc(f, g) = 2 \Pr(f(x) = g(x)) - 1
\]

\[
= 2 \left[ \frac{2^N - hd(f, g)}{2^N} \right] - 1
\]

\[
= \frac{2^{N+1} - 2hd(f, g)}{2^N} - 1
\]

\[
= 1 - \frac{2hd(f, g)}{2^N}
\]

\[
= 1 - \frac{hd(f, g)}{2^{N-1}}
\]

Similarly, let $cc(f, g) = 1 - \frac{hd(f, g)}{2^{N-1}}$. Then

\[
cc(f, g) = 1 - \frac{\Sigma_x (f(x) \oplus g(x))}{2^{N-1}}
\]

\[
= \frac{\Sigma_x 1 - 2\Sigma_x (f(x) \oplus g(x))}{2^N}
\]

\[
= \frac{\Sigma_x [1 - 2(f(x) \oplus g(x))]}{2^N}
\]
Chapter 2. Review of Boolean Function and S-Box Theory

\[
\sum_{x} f(x) \hat{g}(x) = \frac{2^N}{2^N}
\]

A correlation coefficient of zero between two boolean functions, \( f \) and \( g \), would indicate that they are completely uncorrelated, that is, the knowledge of one function does not contribute towards the approximation of the other. In contrast, a value of -1 or 1 for the correlation coefficient means a perfect negative correlation \( (f = \bar{g}) \) or a perfect positive correlation \( (f = g) \) respectively, between the two functions.

Another useful representation of a boolean function is its algebraic normal form (ANF). Every ANF representation corresponds to a unique boolean function truth table.

**Definition 2.6.** The algebraic normal form (ANF) of an \( N \)-variable boolean function, \( f(x) \), is written in the form:

\[
f(x) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_N x_N \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus \ldots \oplus a_{(N-1)N} x_{N-1} x_N \oplus \ldots \oplus a_{123\ldots N} x_1 x_2 x_3 \ldots x_N.
\]

where the coefficients \( a_i \in \{0, 1\} \) form the elements of the truth table of the ANF of \( f(x) \). Note that each product term in the ANF is calculated by the multiplication of each of the components of that term.

**Definition 2.7.** An \( N \)-variable boolean function, \( f(x) \), which contains all \( N \) variables in its ANF is called a nondegenerate function. Conversely, if \( f(x) \) does not contain every variable in its ANF representation then the function is degenerate.

The algebraic degree of a boolean function is a good indicator of the function’s algebraic complexity. The higher the degree of a function, the greater is its algebraic complexity.

**Definition 2.8.** Let \( \text{deg}(f) \) denote the algebraic degree, or simply degree, of a boolean function, \( f(x) \). The algebraic degree (also sometimes referred to as the nonlinear order) is defined to be the number of variables in the largest product term of the function’s ANF having a non-zero coefficient.

**Lemma 2.1.** (From [102]) Let \( f(x) \) be an \( N \)-variable boolean function. Then \( \text{deg}(f) < N \) iff \( 2 \mid \text{hw}(f) \).

Clearly, from the above lemma, all \( N \)-variable boolean functions of degree \( N \) must have odd hamming weight.
Theorem 2.1. (From [11]) Let $f(x)$ be an $N$-variable boolean function and $\text{deg}(f) > 0$. Then $\text{hw}(f)$ is a multiple of $2^{\left\lfloor \frac{N-1}{\text{deg}(f)} \right\rfloor}$.

A subspace of the entire $N$-dimensional boolean function space consists of the affine functions, which include linear functions. These particular functions have an algebraic degree of 1.

Definition 2.9. Let $\omega$ and $x$ be vectors $\in V_2^N$. A boolean function of the form $l_\omega(x) = \omega \cdot x = \omega_1 x_1 \oplus \omega_2 x_2 \oplus ... \oplus \omega_N x_N$ is called a linear function of $N$ variables. This corresponds to the dot product of binary vectors $\omega \in V_2^N$ and $x \in V_2^N$.

The set of $N$-variable affine functions consists entirely of the set of all $N$-variable linear functions and their complements.

Definition 2.10. Let $\omega$ and $x$ be vectors $\in V_2^N$ and $\omega_0 \in V_2$. Then a boolean function of the form $f(x) = \omega_0 \oplus l_\omega(x) = \omega_0 \oplus (\omega \cdot x) = \omega_0 \oplus \omega_1 x_1 \oplus \omega_2 x_2 \oplus ... \oplus \omega_N x_N$ is called an affine function of $N$ variables. Clearly, if $\omega_0 = 0$, then the function is a linear function.

It is useful to consider a transformation of a boolean function which preserves certain cryptographic properties whilst altering others. An affine transformation is such a transformation.

Definition 2.11. An affine transformation on the input of an $N$-variable boolean function, $f(x)$, is defined as the resultant function, $g(x)$, produced by replacing the input vector $x$ with the product of $x$ with the transformation matrix, $A$, and subsequently applying a translation, $b$. Note that $x \in V_2^N$, $A$ is an $N \times N$ non-singular binary matrix, and $b \in V_2^N$. This can be expressed as $g(x) = f(Ax \oplus b)$. If $b = 0$, then this is referred to as a linear transformation. Also, an affine transformation is not invalidated by a translation of the output of the form $g(x) = f(Ax \oplus b) \oplus d$, with $d \in V_2$. Further, an affine transformation can also be achieved by the addition of a boolean function with an affine boolean function.

Thus, the following are all considered to be valid affine transformations:

\[
\begin{align*}
g(x) & = f(Ax) \\
g(x) & = f(Ax) \oplus d \\
g(x) & = f(Ax \oplus b)
\end{align*}
\]
\[ g(x) = f(Ax \oplus b) \oplus d \]
\[ g(x) = f(Ax \oplus b) \oplus c \cdot x \]
\[ g(x) = f(Ax \oplus b) \oplus c \cdot x \oplus d \]

(A an \( N \times N \) non-singular binary matrix, \( b \) and \( c \in V_2^N \) and \( d \in V_2 \)).

Following on from affine transformations, is the relationship of equivalence which allows groupings of boolean functions exhibiting particular combinations of cryptographic properties to be formed. A fundamental observation is that a function and all of its affine transformations belong to the same equivalence class.

**Definition 2.12.** Two distinct \( N \)-variable boolean functions, \( f \) and \( g \), belong to the same equivalence class iff \( \exists \) some \( a \), \( b \), \( c \) and \( M \) such that the following equivalence relation holds:

\[ g(x) = f(Mx \oplus a) \oplus b \cdot x \oplus c \]

(where \( a \) and \( b \) are binary \( N \)-variable vectors, \( c \) is a binary scalar and \( M \) is an \( N \times N \) invertible binary matrix).

An important representation of boolean functions which presents information about a function from a significantly different frame of reference is the Walsh Hadamard transform. As discussed in the next section, this is useful in directly determining certain properties and characteristics of a boolean function.

**Definition 2.13.** The Walsh Hadamard transform (WHT) of the polarity truth table of a boolean function \( \hat{f}(x) \), denoted by \( \hat{F}(\omega) \), is a measure of the correlation between a function and the set of all linear functions. It is defined by

\[
\hat{F}(\omega) = \sum_x (-1)^{f(x)} (-1)^{l_\omega(x)} = \sum_x \hat{f}(x) \hat{l}_\omega(x)
\]

where \( \hat{l}_\omega(x) \in \{1, -1\} \) is the polarity form of the linear function defined by \( \omega \in V_2^N \). Thus, \( \forall \omega, \hat{F}(\omega) \) is a real number in the range \([-2^N, 2^N]\). The vector representing the Walsh Hadamard transform of a function is referred to as its Walsh Hadamard spectrum.

The values in the Walsh Hadamard spectrum of an \( N \)-variable boolean function are constrained by a summative relationship which implicitly limits the magnitude and frequency of those values. This is known as Parseval’s Equation from [60].
Theorem 2.2. Parseval’s Equation ([60]):

\[ \Sigma_\omega \left( \hat{F}(\omega) \right)^2 = 2^{2N} \]

Proof:

\[
\Sigma_\omega \left( \hat{F}(\omega) \right)^2 = \Sigma_\omega (-1)^{f(x)}(-1)^{l_\omega(x)} \Sigma_\omega (-1)^{f(x)}(-1)^{l_\omega(x)} \\
= \Sigma_\omega (-1)^{f(x)}(-1)^{f(x)} \Sigma_\omega (-1)^{l_\omega(x)}(-1)^{l_\omega(x)} \\
= 2^N 2^N \\
= 2^{N+N} \\
= 2^{2N}
\]

□

This value is constant for all \(N\)-variable boolean functions. If a function is boolean, then the values in its Walsh Hadamard spectrum must satisfy Parseval’s Equation. Note however that, even if a function satisfies Parseval’s Equation, it may not necessarily be boolean.

Boolean function concatenation is used to build higher dimensional boolean functions from pairs of lower dimensional functions. In this way, certain characteristics of the component functions can be preserved in the formed function.

Definition 2.14. We denote the concatenation of two \((N-1)\)-variable boolean functions, \(f(x)\) and \(g(x)\), to form an \(N\)-variable boolean function, \(h(x)\), by \(h(x) = f(x) \parallel g(x)\) (or conventionally \(f \parallel g\)). In terms of the algebraic normal form, the concatenation, \(h(x) = f(x) \parallel g(x)\), can be expressed as:

\[
h(x_1, x_2, \ldots, x_N) = (1 \oplus x_N)f(x_1, \ldots, x_{N-1}) \oplus x_N g(x_1, \ldots, x_{N-1})
\]

\[
= f(x) \oplus x_N (f(x) \oplus g(x))
\]

The Walsh Hadamard transform of \(h(x)\), \(\hat{H}(\omega)\), can be found computationally by the following process:

\[
\hat{H}(\omega) = \hat{F}(\omega) + \hat{G}(\omega) \text{ for } \omega \in \{0, \ldots, 2^{N-1} - 1\}
\]

\[
\hat{H}(\omega + 2^{N-1}) = \hat{F}(\omega) - \hat{G}(\omega) \text{ for } \omega \in \{0, \ldots, 2^{N-1} - 1\}
\]

□

It is sometimes necessary to consider the interactions between sets of boolean
functions. To this end, a common approach is to analyze the linear combinations of these sets of functions.

**Definition 2.15.** Let $S$ be the set of $N$-variable boolean functions $\{f_i\}$ where $i = 1, \ldots, M$. Denote by $T$ the set of functions representing the linear combinations of those functions in $S$, which include the set $S$. Then $T$ can be derived by

$$T_j = \bigoplus_{i=1}^{M} (\Gamma f_i)$$

where $j = 1, \ldots, 2^M - 1$ and $\Gamma = \frac{i \& 2^{M-1}}{2^M}$ where $\&$ represents the binary AND operation.

### 2.1.2 Cryptographic Properties of Boolean Functions

The boolean function cryptographic properties relevant to this thesis are defined in this section. We further indicate a means of derivation for these properties and discuss generally how they contribute to the security of boolean functions.

#### Balance

The most basic of all cryptographic properties desired to be exhibited by boolean functions is balance.

**Definition 2.16.** An $N$-variable boolean function, $f(x)$, is said to be balanced if $hw(f) = 2^{N-1}$, or, $\#\{x \mid f(x) = 0\} = \#\{x \mid f(x) = 1\}$. An $N$-variable boolean function whose hamming weight is not equal to $2^{N-1}$ is called an unbalanced function. Clearly, for a balanced function, $f$, $hw(f) = hw(f \oplus 1)$. In terms of its Walsh Hadamard transform, a boolean function is balanced iff $\hat{F}(0) = 0$. In addition, we note that the algebraic degree of an $N$-variable balanced boolean function cannot exceed $N - 1$, as evidenced by Lemma 2.1.

The significance of the balance property is that the higher the magnitude of a function’s imbalance (deviation from $2^{N-1}$), the more likelihood of a high probability linear approximation being obtained. This, in turn, represents a weakness in the boolean function in terms of linear cryptanalysis (see Section 2.3.2). In particular, a large imbalance may enable the boolean function to be easily approximated by a constant function.
Nonlinearity

One of the most important cryptographic properties of a boolean function is its nonlinearity. In effect, the nonlinearity of an $N$-variable boolean function $f$ represents a measure of the dissimilarity between $f$ and the $N$-variable affine function $a$ that $f$ bears the closest bitwise similarity to, measured by the hamming distance between $f$ and $a$. As all affine functions are known to be cryptographically weak, the more dissimilar $f$ is to $a$, the higher its nonlinearity measure and thus the more likely $f$ is able to resist cryptanalytic attacks such as those based on linear approximations. A boolean function with a low nonlinearity value means that the function approaches an affine function and therefore a reasonable approximation by that affine function is probable which, in turn, improves the likelihood of successful linear cryptanalysis (see Section 2.3.2). Thus, it follows that a boolean function which is highly nonlinear is cryptographically desirable as it provides stronger resistance to linear cryptanalysis.

**Definition 2.17.** The nonlinearity of an $N$-variable boolean function $f(x)$, denoted by $NL(f)$, is the minimum distance to the set of all $N$-variable affine functions. Mathematically, the relationship between the nonlinearity of an $N$-variable boolean function $f(x)$ and the Walsh Hadamard transform of that function is given by the following equation:

$$NL(f) = \frac{1}{2}(2^N - WHT_{\text{max}})$$

where $WHT_{\text{max}}$ represents the maximum absolute value in the Walsh Hadamard transform vector.

It is clear that if all values in the Walsh Hadamard spectrum of a boolean function are divisible by 4 then the nonlinearity of the function is even. Similarly, if all values are divisible by 2 but not 4 then the nonlinearity of the function is odd.

The reader should note that a relationship of equality exists between the maximal possible nonlinearity of an $N$-variable boolean function and the covering radius of the first order Reed-Muller Codes $RM(1,N)$ [18].

The nonlinearity of an $N$-variable boolean function is invariant under affine transformations. An affine transformation $g$ on the function $f$ of the form $g(x) = f(Ax \oplus b) \oplus c \cdot x \oplus d$ (with $A$ an $N \times N$ invertible binary matrix; $b$ and $c \in \{0,1\}$).
$V_2^N$ and $d \in V_2$) will not change the nonlinearity of $g$ from that of $f$. The proof requires several lemmas.

**Lemma 2.2.** Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with $g(x) = f(x) \oplus 1 = \overline{f(x)}$. Then $\hat{G}(\omega) = -\hat{F}(\omega)$ for all $\omega \in V_2^N$.

Proof:

\[
\hat{G}(\omega) = \Sigma_x (-1)^{g(x)}(-1)^{l_\omega(x)} = \Sigma_x (-1)^{\overline{f(x)}}(-1)^{l_\omega(x)} = \Sigma_x -(-1)^{f(x)}(-1)^{l_\omega(x)} = -\Sigma_x (-1)^{f(x)}(-1)^{l_\omega(x)} = -\Sigma_x \check{f}(x) \hat{l}_\omega(x) = -\check{F}(\omega)
\]

\hfill \Box

We note from Lemma 2.2 that the complementation of the output of $f(x)$ produces only a change in the sign of its WHT values.

**Lemma 2.3.** [69] Let $A$ be an $N \times N$ invertible binary matrix, and $x$ and $\omega$ be vectors in $V_2^N$. Then the linear function of $Ax$ defined by $\omega$, $l_\omega(Ax) = l_{A^T \omega}(x)$ where $A^T$ is the transpose of the matrix $A$.

Proof:

\[
l_\omega(Ax) = (Ax) \cdot w = (A_{x_1} x_1 \oplus A_{x_2} x_2 \oplus A_{x_3} x_3 \oplus \ldots \oplus A_{x_N} x_N) \omega_1 \oplus (A_{x_1} x_1 \oplus A_{x_2} x_2 \oplus A_{x_3} x_3 \oplus \ldots \oplus A_{x_N} x_N) \omega_2 \oplus \ldots \oplus (A_{x_1} x_1 \oplus A_{x_2} x_2 \oplus A_{x_3} x_3 \oplus \ldots \oplus A_{x_N} x_N) \omega_N = (A_{x_1} \omega_1 \oplus A_{x_2} \omega_2 \oplus A_{x_3} \omega_3 \oplus \ldots \oplus A_{x_N} \omega_N) x_1 \oplus ... \oplus (A_{x_1} \omega_1 \oplus A_{x_2} \omega_2 \oplus A_{x_3} \omega_3 \oplus \ldots \oplus A_{x_N} \omega_N) x_N = (A_{x_1} \omega_1 \oplus A_{x_2} \omega_2 \oplus A_{x_3} \omega_3 \oplus \ldots \oplus A_{x_N} \omega_N) x_N = (A_{x_1} \omega_1 \oplus A_{x_2} \omega_2 \oplus A_{x_3} \omega_3 \oplus \ldots \oplus A_{x_N} \omega_N) x_N = (A^T \omega) \cdot x = l_{A^T \omega}(x)
\]

\hfill \Box

**Lemma 2.4.** [69] Let $f(x)$ be an $N$-variable boolean function, $\check{F}(\omega)$ ($\omega \in V_2^N$) be the Walsh Hadamard transform vector of the polarity truth table of $f$. 

...
and let \( g(x) \) be an \( N \)-variable boolean function with \( g(x) = f(Ax) \) where \( A \) is an \( NxN \) invertible binary matrix. Then the Walsh Hadamard transform of the polarity truth table of \( g(x) \), \( \hat{G}(\omega) = \hat{F}(P\omega) \) where \( P = (A^{-1})^T \).

Proof:

\[
\hat{G}(\omega) = \sum_x (-1)^{g(x)}(-1)^{\omega(x)} = \sum_x (-1)^{f(Ax)}(-1)^{\omega(x)}
\]

Let \( y = Ax \) then \( (A^{-1})y = (A^{-1})Ax \)

\[
(A^{-1})y = x
\]

\[
\hat{G}(\omega) = \sum_y (-1)^{f(y)}(-1)^{\omega((A^{-1})y)} = \sum_y (-1)^{f(y)}(-1)^{\omega((A^{-1})^T\omega(y))} = \sum_y \hat{f}(y)\hat{\omega}(y)
\]

\[
\hat{G}(\omega) = \hat{F}((A^{-1})^T\omega)
\]

From Lemma 2.4, the transforming of the input of \( f \) results in a permutation of the values in the Walsh Hadamard transform vector. More importantly, the magnitude of the Walsh Hadamard transform values remain unchanged.

**Lemma 2.5.** Let \( f(x) \) be an \( N \)-variable boolean function, \( \hat{F}(\omega) (\omega \in V_2^N) \) be the Walsh Hadamard transform vector of the polarity truth table of \( f \), and let \( g(x) \) be an \( N \)-variable boolean function with \( g(x) = f(x \oplus b) \) where \( b \in V_2^N \). Then the Walsh Hadamard transform of the polarity truth table of \( g(x) \), \( \hat{G}(\omega) = \hat{\omega}(b)\hat{F}(\omega) \).

Proof:

\[
\hat{G}(\omega) = \sum_x (-1)^{g(x)}(-1)^{\omega(x)} = \sum_x (-1)^{f(x \oplus b)}(-1)^{\omega(x)}
\]

Replace \( x \) with \( x \oplus b \)

\[
= \sum_x (-1)^{f(x)}(-1)^{\omega(x \oplus b)} = \sum_x (-1)^{f(x)}(-1)^{\omega(x)}(-1)^{\omega(b)} = \sum_x (-1)^{\omega(b)}(-1)^{f(x)}(-1)^{\omega(x)}
\]
\[ \hat{G}(\omega) = \hat{l}_\omega(b)\hat{F}(\omega) \]

From Lemma 2.5, the translation of the input of \( f \) produces a permutation in the values in its Walsh Hadamard transform vector. The magnitude of the Walsh Hadamard transform values are not affected by the translation.

**Lemma 2.6.** Let \( f(x) \) be an \( N \)-variable boolean function, \( \hat{F}(\omega) (\omega \in V_2^N) \) be the Walsh Hadamard transform vector of the polarity truth table of \( f \), and let \( g(x) \) be an \( N \)-variable boolean function with \( g(x) = f(x) \oplus c \cdot x \) \((c \in V_2^N)\). Then the Walsh Hadamard transform of the polarity truth table of \( g(x) \), \( \hat{G}(\omega) = \hat{F}(\omega \oplus c) \).

Proof:

\[
\hat{G}(\omega) = \sum_x (-1)^{g(x)}(-1)^{l_\omega(x)}
\]

\[
= \sum_x (-1)^{f(x)} \oplus c \cdot x (-1)^{l_\omega(x)}
\]

\[
= \sum_x (-1)^{f(x)} (-1)^{c \cdot x}(-1)^{l_\omega(x)}
\]

\[
= \sum_x (-1)^{f(x)} (-1)^{l_\omega(x)} (-1)^{l_\omega(x)}
\]

\[
= \sum_x (-1)^{f(x)} (-1)^{l_\omega(x) \oplus c(x)}
\]

\[
= \hat{F}(\omega \oplus c)
\]

Lemma 2.6 illustrates that a transformation in the output of a function, produces the effect of translating the input of its Walsh Hadamard transform, thus permuting the values but not changing their magnitudes.

**Theorem 2.3.** Let \( f(x) \) be an \( N \)-variable boolean function, \( \hat{F}(\omega) \) be the Walsh Hadamard transform vector of the polarity truth table of \( f \), and let \( g(x) \) be an \( N \)-variable boolean function with \( g(x) = f(Ax \oplus b) \oplus c \cdot x \oplus d \) (with \( A \) an \( NxN \) invertible binary matrix; \( b \) and \( c \in V_2^N \) and \( d \in V_2 \)). Then

\[ NL(g) = NL(f) \]

Proof: This follows from Definition 2.17 and Lemmas 2.2 to 2.6.

Apart from affine transformations as discussed above, there are other means by which boolean functions may be manipulated. The effect of bit complementation in the truth table of a boolean function serves to make a quantifiable change
to its Walsh Hadamard transform vector. This implies that the nonlinearity of the function will also be modified. The magnitude of the change to the Walsh Hadamard transform values is related to the number of bits complemented.

**Lemma 2.7.** Let \( f(x) \) and \( g(x) \) be \( N \)-variable boolean functions with Walsh Hadamard transform vectors \( \hat{F}(\omega) \) and \( \hat{G}(\omega) \) respectively. Let

\[
\begin{align*}
g(x) & = f(i) \oplus 1 \text{ for some } i \in \{0, \ldots, 2^N - 1\} \\
g(x) & = f(x) \{ \forall \ x \mid x = (0, \ldots, 2^N - 1) \} - \{i\}
\end{align*}
\]

Then \( \hat{G}(\omega) = \hat{F}(\omega) + \Delta_{\hat{F}} \) where \( \Delta_{\hat{F}} \in \{-2, +2\} \).

**Proof:**

\[
\begin{align*}
\hat{G}(\omega) & = \Sigma_x (-1)^{g(x)}(-1)^{L(x)} \\
& = \Sigma_x (-1)^{f(x)}(-1)^{L(x)} + (-1)^{f(i)}(-1)^{L(i)} \\
& = \Sigma_x (-1)^{f(x)}(-1)^{L(x)} + \Delta_{\hat{F}} \\
& = \hat{F}(\omega) \pm 2 \\
\end{align*}
\]

where \( \Delta_{\hat{F}} = 2(-1)^{f(i)}(-1)^{L(i)} \)

\( \square \)

**Corollary 2.1.** Let \( f(x) \) and \( g(x) \) be \( N \)-variable boolean functions with nonlinearity \( NL(f) \) and \( NL(g) \) respectively. Let

\[
\begin{align*}
g(x) & = f(i) \oplus 1 \text{ for some } i \in \{0, \ldots, 2^N - 1\} \\
g(x) & = f(x) \{ \forall \ x \mid x = (0, \ldots, 2^N - 1) \} - \{i\}
\end{align*}
\]

Then \( NL(g) = NL(f) \pm 1 \).

**Proof:** This follows from Definition 2.17 and Lemma 2.7. \( \square \)

**Lemma 2.8.** Let \( f(x) \) and \( g(x) \) be \( N \)-variable boolean functions with Walsh Hadamard transform vectors \( \hat{F}(\omega) \) and \( \hat{G}(\omega) \) respectively. Let

\[
\begin{align*}
g(x) & = f(i) \oplus 1 \text{ for some } i \in \{0, \ldots, 2^N - 1\} \text{ where } f(i) = 0 \\
g(x) & = f(j) \oplus 1 \text{ for some } j \in \{0, \ldots, 2^N - 1\} \text{ where } f(j) = 1 \text{ and } i \neq j \\
g(x) & = f(x) \{ \forall \ x \mid x = (0, \ldots, 2^N - 1) \} - \{i\} - \{j\}
\end{align*}
\]
Then $\hat{G}(\omega) = \hat{F}(\omega) + \Delta_{\hat{F}}$ where $\Delta_{\hat{F}} \in \{-4, 0, +4\}$.

Proof:

\[
\hat{G}(\omega) = \sum_x (-1)^{g(x)}(-1)^{\omega(x)} \\
= \sum_{x \neq i, x \neq j} (-1)^{f(x)}(-1)^{\omega(x)} + (-1)^{f(i)}(-1)^{\omega(i)} + (-1)^{f(j)}(-1)^{\omega(j)} \\
= \sum_x (-1)^{f(x)}(-1)^{\omega(x)} + \Delta_{\hat{F}} \\
= \hat{F}(\omega) \pm \{0, 4\}
\]

where $\Delta_{\hat{F}} = 2(-1)^{f(i)} = 0(-1)^{\omega(i)} + 2(-1)^{f(j)} = 1(-1)^{\omega(j)}$

\[\square\]

Corollary 2.2. Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with nonlinearity $NL(f)$ and $NL(g)$ respectively. Let

\[
g(x) = f(i) \oplus 1 \quad \text{for some } i \in \{0, \ldots, 2^N - 1\} \quad \text{where } f(i) = 0 \\

\]

\[
g(x) = f(j) \oplus 1 \quad \text{for some } j \in \{0, \ldots, 2^N - 1\} \quad \text{where } f(j) = 1 \quad \text{and } i \neq j \\

\]

\[
g(x) = f(x) \{ \forall x \mid x = (0, \ldots, 2^N - 1) \} - \{i\} - \{j\}
\]

Then $NL(g) = NL(f) + \Delta_{NL(f)}$ where $\Delta_{NL(f)} \in \{0, +2, -2\}$

Proof: This follows from Definition 2.17 and Lemma 2.8. \[\square\]

As the process of complementing bits results in defined changes to the non-linearity of a boolean function, this technique often forms part of heuristic algorithms for the generation of highly nonlinear boolean functions.

Avalanche

The concept of an avalanche effect was first identified by Feistel [29] as a notion that the spreading of the input bits throughout a process will result in a likely uniform distribution in the number of 0 and 1 bits in the output due to a strong output dependence on all of the input bits. This notion has since been refined to form the basis of a number of cryptographic properties of cipher systems and components, and in particular of boolean functions.

We now state the well-known definition of the derivative of a boolean function [50]. The boolean function derivative is required to assist in the measure of a function’s avalanche characteristics.

Definition 2.18. Let $d_\alpha f(x)$ denote the boolean function derivative of an
2.1. Boolean Function Theory

N-variable boolean function $f(x)$ in the direction $\alpha$, $\alpha \in V_2^N$. Then

$$d_\alpha f(x) = f(x) \oplus f(x \oplus \alpha)$$

$$= \hat{f}(x) \hat{f}(x \oplus \alpha)$$

The derivative of a boolean function gives a measure of the effect on the output of a function when there is a change in the input in the direction $\alpha$. In other words, the derivative provides the function output difference given a particular input difference.

**Definition 2.19.** Denote the autocorrelation function of $\hat{f}(x)$ (the polarity truth table of an $N$-variable boolean function $f(x)$) by $\hat{r}_f(\alpha)$. This function may be calculated as follows:

$$\hat{r}_f(\alpha) = \Sigma_x \hat{f}(x) \hat{f}(x \oplus \alpha)$$

with $\alpha$ and $x \in \{0,\ldots,2^N - 1\}$ and $\hat{r}_f(0) = 2^N$. □

The autocorrelation function measures the directional derivative of a function for an input shift in the direction of $\alpha$ over all $x \in V_2^N$. Note that the autocorrelation function sums the polarity form of the boolean function derivative. The autocorrelation function of an $N$-variable boolean function, also denoted by $\text{AC}$, is a real-valued vector containing the $2^N$ values of $\hat{r}(\alpha)$ for each $\alpha = 0,\ldots,2^N - 1$. The range of these values is $[-2^N, 2^N]$.

Two important values [105] used to ascertain the quality of the avalanche characteristics exhibited by a function may be obtained from the autocorrelation function. These are the absolute indicator and the sum-of-square indicator.

**Corollary 2.3.** Let $|\text{AC}|_{\text{max}}$ denote the absolute indicator (or maximum absolute autocorrelation value) derived from the autocorrelation function $\hat{r}(\alpha)$. Then

$$|\text{AC}|_{\text{max}} = \max_{\alpha} |\hat{r}(\alpha)|$$

with $\alpha \in \{1,\ldots,2^N - 1\}$ □

Hereinafter we shall simply use $\text{AC}_{\text{max}}$ to refer to the absolute indicator (or maximum absolute autocorrelation value).

**Corollary 2.4.** Let $\sigma$ denote the sum-of-square indicator, also derived from
the autocorrelation function $\hat{r}(\alpha)$. Then

$$\sigma = \sum_{\alpha} \hat{r}^2(\alpha)$$

with $\alpha \in \{0, \ldots, 2^N - 1\}$.

The above autocorrelation measures of a boolean function are invariant under affine transformations. An affine transformation $g$ on an $N$-variable boolean function $f$ of the form $g(x) = f(Ax \oplus b) \oplus c \cdot x \oplus d$ (with $A$ an $N \times N$ invertible binary matrix; $b$ and $c \in V_2^N$ and $d \in V_2$) will not change the maximum absolute autocorrelation value nor the sum-of-square indicator of $g$ from that of $f$.

**Lemma 2.9.** Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with autocorrelation functions of polarity forms $\hat{r}_f(\alpha)$ and $\hat{r}_g(\alpha)$ ($\alpha \in V_2^N$) respectively. Let $g(x) = f(Ax)$ where $A$ is an $N \times N$ invertible binary matrix. Then $\hat{r}_g(\alpha) = \hat{r}_f(A\alpha)$.

Proof:

$$\hat{r}_g(\alpha) = \sum_x \hat{g}(x) \hat{g}(x \oplus \alpha)$$

$$= \sum_x \hat{f}(Ax) \hat{f}(A(x \oplus \alpha))$$

$$= \sum_x \hat{f}(Ax) \hat{f}(Ax \oplus A\alpha)$$

Let $y = Ax$

$$= \sum_y \hat{f}(y) \hat{f}(y \oplus A\alpha)$$

$$\hat{r}_g(\alpha) = \hat{r}_f(A\alpha)$$

From Lemma 2.9, transforming the input of $f$ results in a permutation of the values in its autocorrelation vector. More importantly, the magnitude of the autocorrelation values remain unchanged.

**Lemma 2.10.** Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with autocorrelation functions of polarity forms $\hat{r}_f(\alpha)$ and $\hat{r}_g(\alpha)$ ($\alpha \in V_2^N$) respectively. Let $g(x) = f(x \oplus b)$ where $b \in V_2^N$. Then $\hat{r}_g(\alpha) = \hat{r}_f(\alpha)$.

Proof:

$$\hat{r}_g(\alpha) = \sum_x \hat{g}(x) \hat{g}(x \oplus \alpha)$$

$$= \sum_x \hat{f}(x \oplus b) \hat{f}(x \oplus b \oplus \alpha))$$

$$= \sum_x \hat{f}(x) \hat{f}(x \oplus \alpha))$$
From Lemma 2.10, the translation of the input of $f$ produces no change in the values of its autocorrelation vector.

Lemma 2.11. Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with autocorrelation functions of polarity forms $\hat{r}_f(\alpha)$ and $\hat{r}_g(\alpha)$ ($\alpha \in V_2^N$) respectively. Let $g(x) = f(x) \oplus c \cdot x$ ($c \in V_2^N$). Then $\hat{r}_g(\alpha) = \hat{l}_c(\alpha)\hat{r}_f(\alpha)$.

Proof:

\[
\hat{r}_g(\alpha) = \Sigma_x g(x)\hat{g}(x \oplus \alpha)
= \Sigma_x \hat{f}(x)(-1)^{c(x)f(x \oplus \alpha)}(-1)^{c(x \oplus \alpha)}
= \Sigma_x \hat{f}(x)\hat{l}_c(x)\hat{f}(x \oplus \alpha)\hat{l}_c(x \oplus \alpha)
= \Sigma_x \hat{l}_c(x)\hat{l}_c(x \oplus \alpha)\hat{f}(x)\hat{f}(x \oplus \alpha)
= \hat{l}_c(\alpha)\Sigma_x \hat{f}(x)\hat{f}(x \oplus \alpha)
= \hat{l}_c(\alpha)\hat{r}_f(\alpha)
\]

Lemma 2.11 illustrates that a transformation in the output of a function, produces the effect of changing the signs of some of the values in the autocorrelation vector. This does not, however, change the magnitude of these values.

Lemma 2.12. Let $f(x)$ and $g(x)$ be $N$-variable boolean functions with autocorrelation functions of polarity forms $\hat{r}_f(\alpha)$ and $\hat{r}_g(\alpha)$ ($\alpha \in V_2^N$) respectively. Let $g(x) = f(x) \oplus 1 = \overline{f(x)}$. Then $\hat{r}_g(\alpha) = \hat{r}_f(\alpha)$ \(\forall \alpha \in V_2^N\).

Proof:

\[
\hat{r}_g(\alpha) = \Sigma_x g(x)\hat{g}(x \oplus \alpha)
= \Sigma_x (-1)^{f(x)-1}\overline{f(x \oplus \alpha)}
= \Sigma_x (-1)^f(x) - (-1)^f(x \oplus \alpha)
= \Sigma_x (-1)^f(x)(-1)^f(x \oplus \alpha)
= \Sigma_x \hat{f}(x)\hat{f}(x \oplus \alpha)
= \hat{r}_f(\alpha)
\]

From Lemma 2.12, complementing the output of a boolean function leaves
the autocorrelation vector unchanged.

**Theorem 2.4.**  Let \( f(x) \) and \( g(x) \) be \( N \)-variable boolean functions with absolute indicators \( AC_{max}(\hat{f}) \) and \( AC_{max}(\hat{g}) \) and sum-of-square indicators \( \sigma(\hat{f}) \) and \( \sigma(\hat{g}) \) respectively. Let \( g(x) = f(Ax \oplus b) \oplus c \cdot x \oplus d \) (with \( A \) an \( N \times N \) invertible binary matrix; \( b \) and \( c \in V_2^N \) and \( d \in V_2^N \)). Then

\[
AC_{max}(\hat{f}) = AC_{max}(\hat{g})
\]

and

\[
\sigma(\hat{f}) = \sigma(\hat{g})
\]

**Proof:** This follows from Definition 2.19, Corollaries 2.3 and 2.4, and Lemmas 2.9 to 2.12.

The overall effect of an affine transformation on the autocorrelation vector of a boolean function can be seen by combining the operations performed in Lemmas 2.9 to 2.12. The magnitudes remain constant, however, the signs and sequence of the autocorrelation values may vary. Thus, the absolute indicator and sum-of-square indicator of a boolean function are unaffected by affine transformation.

The concept of Global Avalanche Criteria (GAC) was first proposed by Zhang and Zheng [105] whereby the overall avalanche characteristic of a boolean function is evaluated with the use of its absolute indicator (or maximum absolute autocorrelation value), as well as its sum-of-square indicator.

**Definition 2.20.**  An \( N \)-variable boolean function, \( f(x) \), is said to contain at least one non-zero linear structure if \( \hat{r}_j(\alpha) = \pm 2^N \) for some non-zero \( \alpha \in V_2^N \).

Following on from Definition 2.20, for some input change, \( \alpha \), either the output of the function \( f(x) \) does not change at all (the case where \( \hat{r}(\alpha) = 2^N \)), or is completely complemented (the case where \( \hat{r}(\alpha) = -2^N \)). The autocorrelation vector of an affine function is such that \( \hat{r}(\alpha) \) is equal to \( \pm 2^N \) for all possible values of \( \alpha \in V_2^N \).

An important boolean function property known as the Strict Avalanche Criterion (SAC) was first introduced by Webster and Tavares [101]. This property can be defined in terms of the truth table form of the directional derivative of a function or by using the polarity form of the directional derivative by way of its autocorrelation function.
Definition 2.21. An $N$-variable boolean function $f(x)$ is said to satisfy strict avalanche criterion ($SAC$) if, for every $s$ such that $hw(s) = 1$, $\sum_x f(x) \oplus f(x \oplus s) = 2^{N-1}$. Alternatively, the strict avalanche criterion is said to be satisfied for a boolean function, $f(x)$, if the autocorrelation function of $\hat{f}(x)$, $\hat{r}_f(s)$, contains zero values in all positions $s$ where $hw(s) = 1$.

The satisfaction of the strict avalanche criterion means that the function’s derivative over all inputs and for all values of the input shift, $s$, with weight 1, is balanced. This implies therefore that the output difference for all single bit changes to the input must be uniformly distributed.

In 1990, Preneel et al. extended the earlier concept of the Strict Avalanche Criterion to the more general measure of Propagation Criteria in [79].

Definition 2.22. An $N$-variable boolean function, $f(x)$, is said to satisfy the propagation criteria of degree $k$, $PC(k)$, with respect to a non-zero vector $\alpha \in V_2^N$ if

$$\sum_x f(x) \oplus f(x \oplus \alpha) = 2^{N-1}$$

$\forall \alpha$ such that $1 \leq hw(\alpha) \leq k$.

Thus, a boolean function is said to satisfy $PC(k)$ if when any $t$ input bits ($1 \leq t \leq k$) are complemented, this results in a change in each output with probability $1/2$. Propagation criteria of degree 1, $PC(1)$, is equivalent to the Strict Avalanche Criterion ($SAC$).

The autocorrelation vector over all $\alpha \in \{0,\ldots,2^N-1\}$ provides a boolean function’s highest possible degree of propagation criteria, $k$, from elements which are zero valued for all $\alpha$ satisfying $1 \leq hw(\alpha) \leq k$. Thus, there is a relationship of equality between $PC(k)$ and a zero valued autocorrelation function, $\hat{r}(\alpha)$, for all $\alpha$ satisfying $1 \leq hw(\alpha) \leq k$.

Definition 2.23. An $N$-variable boolean function, $f(x)$, is said to satisfy the propagation criteria of degree $k$ and order $r$, ie $PC(k)$ of order $r$, with respect to a non-zero vector $\alpha \in V_2^N$ if

$$\sum_x f(x) \oplus f(x \oplus \alpha) = 2^{N-1}$$

if any $r$ bits of $x$ are held constant, and $1 \leq hw(\alpha) \leq k \forall \alpha$ where $k + r \leq N$.

For a definition of Extended $PC(k)$ of order $r$, see [79]. Note that [48] defined
for boolean functions the concept of $\epsilon$-almost $PC(k)$ of order $r$. The reader is referred to [48] for more details.

The propagation criteria property is not invariant under a linear or affine transformation. Consider the linear transformation $g$ on the function $f$ in Lemma 2.9. The transforming of the input of $f$ results in a permutation of the values in the autocorrelation vector. It is clear from Definitions 2.21 and 2.22, and Lemma 2.9 that such a permutation can change the degree of propagation criteria of a boolean function.

As previously stated, the Global Avalanche Criteria ($GAC$) can be evaluated using the absolute indicator, $AC_{\text{max}}$, and the sum-of-square indicator, $\sigma$. An important boolean function property which contributes to the avalanche characteristics of a function is propagation criteria. The satisfaction of $PC(k)$ clearly provides a number of zero values in the autocorrelation vector and thus reduces the function’s sum-of-square indicator. Another important feature of boolean functions for good avalanche is the absence of non-zero linear structures. As linear structures set values of magnitude $2^N$ in the autocorrelation vector of a function, minimizing the number of non-zero linear structures has the effect of reducing the overall sum-of-square indicator. Eliminating non-zero linear structures altogether has the added effect of ensuring that the absolute indicator $< 2^N$. Lower values of each of the two $GAC$ measures indicate better avalanche characteristics of the function.

The autocorrelation function provides a useful means of measuring the two indicators of which minimizations are sought in order to improve avalanche characteristics of a function. Propagation Criteria is a boolean function property which enables a function to achieve good diffusion through ensuring output uniformity under changes to the function input obtained through strong output dependence on all of the input bits. Cipher systems which employ boolean functions with good avalanche characteristics reduce their vulnerability to differential attacks (see Section 2.3.1).

**Correlation Immunity**

Correlation immunity is a property of boolean functions which denotes the extent of independence between linear combinations of the input bits and the output.

The following definition is due to [34].

**Definition 2.24.** An $N$-variable boolean function, $f(x)$, is $m^{th}$-order corre-
lution immune, denoted $CI(m)$, if it is statistically independent of the subset of $m$ input variables where $1 \leq m \leq N$.

Alternatively, a function’s order of correlation immunity may be determined using the relationship between its Walsh Hadamard transform and the hamming weights of its inputs.

**Definition 2.25.** An $N$-variable boolean function, $f(x)$, is $m^{th}$-order correlation immune, denoted by $CI(m)$, if, for every $\omega$ such that $1 \leq hw(\omega) \leq m$, $\hat{F}(\omega) = 0$.

**Definition 2.26.** An $N$-variable boolean function, $f(x)$, which is both balanced and $m^{th}$-order correlation immune, is known as an $m$-resilient boolean function.

The well known inequality due to Siegenthaler [95] gives an upper bound on the algebraic degree of a boolean function $f(x)$, $deg(f)$, as it relates to the number of input variables, $N$, and the order of correlation immunity, $m$.

**Theorem 2.5.** (From [95]) Siegenthaler’s Inequality:

$$N \geq m + deg(f) + \epsilon \quad \text{where} \quad \epsilon = \begin{cases} 0 & \text{if function is balanced} \\ 1 & \text{if function is unbalanced} \end{cases}$$

Proof: Refer to [95].

The correlation immunity property is not invariant under a linear or affine transformation. Consider the linear transformation $g$ on the function $f$ in Lemma 2.4. The transforming of the input of $f$ results in a permutation of the values in the Walsh Hadamard transform vector. It is clear from Definition 2.25 and Lemma 2.4 that such a permutation can alter a function’s order of correlation immunity.

A function’s order of correlation immunity may also be altered by bit complementation on a boolean function truth table. This can be seen from Definition 2.25, and Lemmas 2.7 and 2.8 in that Walsh Hadamard transform values may change from non-zero values to zero values or vice versa in positions where the hamming weights of $\omega$ dictate a change in the order of correlation immunity.

The following theorem from [86] provides a divisibility result of significance in the analysis of resilient boolean functions.

**Theorem 2.6.** (From [86]) Let $f(x)$ be an $m$-resilient boolean function of $N$ variables. Then $\hat{F}(\omega) \equiv 0 \mod 2^{m+2}$.

Note that [46] defined the concept of an $\epsilon$-almost $k$-resilient boolean function.
The reader is referred to [46] for more details.

The order of correlation immunity (and also resilience) represents the absence of any dependence between the output of a function and a fixed subset of input bits, that subset being larger if the order of correlation immunity is high. Correlation attacks (see Section 2.3.3) exploit any such dependence which may exist within combining functions of stream ciphers which are directly involved in producing the keystream.

2.1.3 Relationships Between Cryptographic Properties of Boolean Functions

In an ideal situation, a combination of a large number of desirable cryptographic properties such as those discussed above, all with adequate measures, would be exhibited by individual boolean functions. In reality, such functions cannot exist given the inter-relationships between certain cryptographic properties and the strict rules which limit boolean function characteristics. We now comment on specific inter-relations between pairs of properties and discuss how these properties affect each other.

Nonlinearity and Avalanche

From previous discussions, recall that the nonlinearity property is an expression of the dissimilarity between a function and the nearest affine function in the set. Nonlinearity is often calculated through applying the Walsh Hadamard transform of a boolean function. The avalanche characteristic is the quality which describes how well input propagates throughout a process and affects the output uniformly. The autocorrelation function is used to measure the avalanche characteristics of a boolean function and determines the two important indicators, maximum absolute autocorrelation value (absolute indicator) and sum-of-square indicator. The propagation criteria property quantifies the level at which a function is able to exhibit uniformity of output differences given specified input differences. The following theorems will assist in establishing the relationship between nonlinearity and avalanche in terms of how they affect each other when they are exhibited together in a boolean function.

The well-known Wiener-Khintchine theorem [93] provides a direct link between the autocorrelation vector and the Walsh Hadamard transform of a func-
2.1. Boolean Function Theory

The theorem is stated here as follows:

**Theorem 2.7.** Let \( f(x) \) be an \( N \)-variable boolean function with the Walsh Hadamard transform of its polarity truth table being \( \hat{F}(\omega) \) and autocorrelation function \( \hat{r}_f(\alpha) \). Then

\[
\Sigma_\alpha \hat{r}_f(\alpha) \hat{l}_\alpha(\omega) = (\hat{F}(\omega))^2
\]

where \( \hat{l}_\alpha(\omega) \) is the polarity form of the linear function of \( \omega \) defined by \( \alpha \).

Proof:

\[
\Sigma_\alpha \hat{r}_f(\alpha) \hat{l}_\alpha(\omega) = \Sigma_\alpha \left[ \sum_x \hat{f}(x) \hat{\bar{f}}(x \oplus \alpha) \hat{l}_\alpha(\omega) \right]
= \Sigma_\alpha \sum_x (-1)^{f(x)}(-1)^{f(x \oplus \alpha)}(-1)^{\alpha \cdot \omega}
\]

Let \( y = x \oplus \alpha \)

\[
= \Sigma_y \sum_x (-1)^{f(x)}(-1)^{f(y)}(-1)^{(x \oplus y) \cdot \omega}
= \Sigma_y \sum_x (-1)^{f(x)}(-1)^{f(y)} \hat{l}_\omega(x) \hat{l}_\omega(y)
= \Sigma_x (-1)^{f(x)} \hat{l}_\omega(x) \Sigma_y (-1)^{f(y)} \hat{l}_\omega(y)
= \Sigma_x \hat{f}(x) \hat{l}_\omega(x) \Sigma_x \hat{\bar{f}}(x) \hat{l}_\omega(x)
= \left( \hat{F}(\omega) \right)^2
\]

\( \square \)

It is clear from Corollary 2.4 that if the sum-of-square indicator, \( \sigma \), is large then \( \hat{r}(\alpha) \) will contain values of large magnitude. Thus, from Theorem 2.7 we can deduce that if \( \sigma \) is large then the nonlinearity of the function is likely to be low. If \( \sigma \) is small, however, then \( NL(f) \) may be high.

In 2003, Zheng and Zhang [114] presented the following theorem relating the nonlinearity of a boolean function with its degree of propagation criteria by specifying a lower bound.

**Theorem 2.8.** (From [114]) Let \( f(x) \) be an \( N \)-variable boolean function satisfying \( PC(k) \). Then

(i) \( NL(f) \geq 2^{N-1} - 2^{N-1-\frac{k}{2}} \),

(ii) From (i), \( NL(f) = 2^{N-1} - 2^{N-1-\frac{k}{2}} \) iff either:

(a) \( N \) is odd, \( k = N - 1 \), and \( f(x) \) is of the form

\[
f(x) = g(x_1 \oplus x_N, \ldots, x_{N-1} \oplus x_N) \oplus h(x_1, \ldots, x_N)
\]
where \( g \) is an \((N-1)\)-variable bent function and \( h \) is an \( N \)-variable affine function; OR

\( \text{OR} \)

(b) \( N \) is even, \( k = N \), and \( f(x) \) is a bent function.

Proof: Refer to [114]. \( \square \)

The above theorem demonstrates that the greater the degree of propagation criteria, \( k \), the greater the minimum nonlinearity of the function will be, and therefore higher nonlinearities are possible.

Some interesting work in [107] showed that, for large \( N \), where conventional nonlinearity measures become infeasible, the nonlinearity of a function may be estimated using information about the autocorrelation function. Theorem 10 from [107] presents a tight upper bound on nonlinearity:

**Theorem 2.9.**  (From [107]) Let \( f(x) \) be an \( N \)-variable boolean function with autocorrelation function \( \hat{r}_f(\alpha) \) and nonlinearity, \( NL(f) \). Then \( NL(f) \) satisfies

\[
NL(f) \leq 2^{N-1} - \frac{1}{2} \sqrt{2^N + AC_{max}}
\]

where \( AC_{max} = max_{\alpha} |\hat{r}_f(\alpha)|. \)

Proof: Refer to [107]. \( \square \)

Clearly, if \( f(x) \) is a bent function (See Section 2.1.4) ie \( AC_{max} = 0 \), the equality of the above theorem holds. In other cases, as \( AC_{max} \) increases the upper bound on nonlinearity decreases.

**Theorem 2.10.**  (From [6]) Let \( f(x) \) be an \( N \)-variable boolean function with sum-of-square indicator, \( \sigma(\hat{f}) \), and maximum absolute Walsh Hadamard transform value, \( WHT_{max} \). Then

\[
\sigma(\hat{f}) \leq 2^N (WHT_{max})^2
\]

Proof: Refer to [6]. \( \square \)

From Theorem 2.10, it can be observed that the higher the nonlinearity of the function (that is, lower \( WHT_{max} \)), the lower the upper bound on the function’s sum-of-square indicator will be.

These four theorems provide evidence that a clear relationship exists between nonlinearity and avalanche. Nonlinearity and avalanche complement each other in a function, ie optimizing one allows the other to be improved. Other research work has been done in investigating this relationship, for example, ([93], [109], [94]).
Nonlinearity and Correlation Immunity

The relationship between the two cryptographic properties of nonlinearity and correlation immunity can be expressed through a study of the effect of Parseval’s Equation (Theorem 2.2). Recall that to achieve an order $m$ of correlation immunity, the values in the Walsh Hadamard transform vector $\hat{F}(\omega)$ corresponding to positions $hw(\omega) \leq m$ must be zero (Definition 2.25). Thus, the higher the order of correlation immunity $m$, the more positions in $\hat{F}(\omega)$ must have values of zero. It then follows that for Parseval’s Equation to remain valid, the higher the magnitude of the non-zero values in $\hat{F}$ must be, which results in a lower nonlinearity. Conversely, the higher the nonlinearity of a function, the lower the magnitude of the non-zero values in its Walsh Hadamard transform vector (particularly the $WHT_{max}$ value). Therefore, in order to satisfy Parseval’s Equation, the number of zero positions must be fewer which means that only a low order of correlation immunity will be possible.

An examination of Siegenthaler’s Inequality (Theorem 2.5) also enables a relationship between the algebraic degree of a boolean function and its order of correlation immunity to be drawn. Provided that the dimension of the boolean function $N$ remains fixed, there exists an opposite relation between the order of correlation immunity and algebraic degree when one of these measures is high. The higher the order of correlation immunity, the lower the algebraic degree of that function must be. Conversely, if the order of correlation immunity is low then the algebraic degree of the function may be high.

Siegenthaler’s Inequality is extended by Carlet in [11] to identify the subsets of values from which the non-zero elements of the Walsh Hadamard transform vector can be taken.

**Theorem 2.11.** (From [11]) Let $f(x)$ be an $m$-resilient boolean function of $N$ variables with $0 \leq m \leq N - 2$, and let $deg(f)$ be the algebraic degree of the function, $f(x)$. Then for every $N$-variable affine function, $a$, $hd(f, a)$ is divisible by $2^{m+1+[\frac{N-m-2}{deg(f)}]}$.

Proof: Refer to [11].

The above theorem provides information about the divisibility of the possible nonlinearity value for $f(x)$, as a value dependent on the function’s number of input variables, order of correlation immunity and algebraic degree. This has
been further elaborated on in [13]. Thus, for some fixed \( m \), if the algebraic degree of \( f(x) \) is low then the nonlinearity has large divisors causing the interval between successive valid nonlinearity values to be large. From this we may infer, particularly for even \( N \), that the highest nonlinearity achievable by a resilient function \( f(x) \) having low algebraic degree will be suboptimal given that it will be at least one large interval below bent nonlinearity (see Section 2.1.4). The same argument holds when the order of correlation immunity, \( m \), is high. The small divisors that result when algebraic degree is higher and \( m \) low provide the possibility of valid nonlinearity values closer to bent nonlinearity, when \( N \) is even.

An upper bound on the nonlinearity of an \( m \)-resilient function was proposed independently in [112], [98], [100] and [84] as follows:

**Theorem 2.12.** Let \( f(x) \) be an \( m \)-resilient boolean function of \( N \) variables with \( 0 \leq m \leq N - 2 \), and with nonlinearity \( NL(f) \). Then, \( NL(f) \leq 2^{N-1} - 2^{m+1} \).

\[ \square \]

It can be seen from Theorem 2.12 and the preceding discussion, that nonlinearity and correlation immunity are opposing properties. Optimizing nonlinearity results in a lower achievable order of correlation immunity. Enforcing a higher order of correlation immunity reduces the maximum achievable nonlinearity.

**Correlation Immunity and Avalanche**

We now briefly discuss the relationship between the correlation immunity property and avalanche, specifically in terms of propagation criteria, and one of the two indicators of \( GAC \), \( AC_{max} \).

The following theorems from [113] provide a lower bound on the absolute indicator (maximum absolute autocorrelation value) of \( CI(m) \) boolean functions, both balanced and unbalanced.

**Theorem 2.13.** (From [113]) Let \( f(x) \) be an \( m^{th} \)-order correlation immune boolean function of \( N \) variables with \( 2 \leq m \leq N \) and maximum absolute autocorrelation value \( AC_{max} \). Then

\[ AC_{max} \geq 2^{m-1} \sum_{i=0}^{\infty} 2^{i(m-1-N)} \]

Proof: Refer to [113]. \( \square \)

**Theorem 2.14.** (From [113]) Let \( f(x) \) be an \( m \)-resilient boolean function of \( N \) variables with \( 1 \leq m \leq N - 1 \) and maximum absolute autocorrelation value
Then

\[ AC_{\text{max}} \geq 2^m \sum_{i=0}^{+\infty} 2^{i(m-N)} \]

Proof: Refer to [113].

Theorems 2.13 and 2.14 illustrate the effect of an increase in \( m \) on the magnitude of \( AC_{\text{max}} \). As the order of correlation immunity increases, the summation on the right hand side of each inequality tends to 2, and \( AC_{\text{max}} \) tends to \( 2^N \). Recall that \( AC_{\text{max}} = 2^N \) for all affine functions and functions with non-zero linear structures.

**Theorem 2.15.** (From [114]) Let \( f(x) \) be an \( m \)-resilient boolean function of \( N \) variables which satisfies \( PC(k) \). Then \( m + k \leq (N - 2) \).

Proof: Refer to [114].

It can be established from Theorem 2.15 that for some fixed \( N \), the higher the order of resilience the lower the degree of propagation criteria of a boolean function. Conversely, the higher the degree of propagation criteria, the lower must be the order of resilience.

The relationship between correlation immunity and avalanche, as it relates to their being exhibited together by a boolean function, has been shown to be in conflict in that a distinct tradeoff exists and what is optimal in one must be least favourable for the other.

### 2.1.4 Some Special Boolean Functions

We now discuss three special types of boolean functions, and the identifying properties which enable them to be classified as such.

**Bent Functions**

A particular class of boolean functions exhibiting unique characteristics was first reported by Rothaus in [82]. These functions were referred to in that paper as “bent functions”. Bent functions were later called “perfect nonlinear” [62] in light of their optimal distance to linear structures.

Bent functions exist only in the space of even dimensional boolean functions. It is not possible for a boolean function to exist that satisfies all the necessary characteristics to be considered bent when the space is odd dimensional.
The Walsh Hadamard spectrum of a bent function is a two-valued spectrum and consists entirely of $\pm 2^{N/2}$ values. Thus, the Walsh Hadamard spectrum is flat. It follows by definition that the nonlinearity of a bent function will be $2^{N-2^{N/2}}$. As Parseval’s Theorem must hold, clearly this is the maximum achievable nonlinearity for $N$-dimensional boolean functions ($N$ even). This indicates that a bent function is at maximum distance from the set of all affine functions. Bent functions also have maximum distance to linear structures. Further, as there can be no zero-valued entries in the Walsh Hadamard spectrum, bent functions do not exhibit any order of correlation immunity.

The autocorrelation vector of an $N$-variable bent boolean function ($N$ even) takes the form $\hat{r}(\alpha) = \{2^N, 0, 0, \ldots, 0\}$. The first entry always has the value $2^N$ and all other entries are 0. Thus, bent functions satisfy propagation criteria of degree $N$, $PC(N)$, and exhibit perfect diffusion with respect to output uniformity given shifts in the input of a bent function.

Although bent functions exhibit cryptographically optimal properties in terms of maximal nonlinearity and perfect (minimal) autocorrelation, $N$-variable bent functions have a hamming weight of $2^{N-1} \pm 2^{N/2-1}$. This indicates a bias from balance of constant magnitude $2^{N/2-1}$; bent functions are never balanced. Furthermore, all $N$-variable bent functions have algebraic degree $\leq \frac{N}{2}$ for $N > 2$. Thus, these characteristics of being unbalanced and having low algebraic degree are cryptographically undesirable for bent functions to be of direct practical use.

Various techniques for the construction of bent functions have been proposed in the literature. Some examples include [59], [71], [90], [27] and [9].

**Semi-Bent Functions**

The cryptographic limitations of bent functions discussed above (unbalanced, low algebraic degree) prevent bent functions from being useful cryptographically. Semi-bent functions were introduced by Chee et al. in [14]. These semi-bent functions attempt to retain the desirable characteristics of bent functions, namely high nonlinearity and zero autocorrelation, whilst ensuring balance.

A semi-bent function, $f(x)$, is an odd-dimensional boolean function constructed by concatenating a bent function, $g(x)$, to the same bent function, $g(x)$, that has had an affine transformation applied to its input and its output complemented.

**Definition 2.27.** (From [14]) Let $f(x)$ be an $N$-variable semi-bent boolean function.
function \( (N \text{ odd}) \) and \( g(x) \) be an \((N-1)\)-variable bent function. Then \( f(x) \) is of the form

\[
f(x) = g(x) \| (g(Ax + b) \oplus 1)
\]

A semi-bent function, \( f(x) \), constructed in this manner is always balanced.

The nonlinearity of an \( N \)-variable semi-bent function is \( 2^{N-1} - 2^{\frac{N-1}{2}} \).

The correlation coefficients (see Definition 2.5) between an \( N \)-variable semi-bent function \( (N \text{ odd}) \) and the set of all \( N \)-variable linear functions always take one of the values in the set \( \{0, \pm 2^{\frac{1-N}{2}}\} \). For an \( N \)-variable semi-bent function, \( f(x) \), \(#(cc(f,l) = 0) = 2^{N-1} \) and \(#(cc(f,l) = \pm 2^{\frac{1-N}{2}}) = 2^{N-1} \forall l \) in the set of \( N \)-variable linear boolean functions. The former represents no correlation between \( f(x) \) and half of all \( N \)-variable linear functions. The latter indicates uniform correlation to the other half of the set of all linear functions.

An \( N \)-variable semi-bent function \( (N \text{ odd}) \) of degree \((N-1)\) also satisfies propagation criteria of degree \( N \), \( PC(N) \). The Strict Uncorrelated Criterion as introduced and defined by Chee et al. in [14] may also be satisfied by pairs of semi-bent functions under certain conditions.

Thus, semi-bent functions represent a useful grouping of odd-dimensional boolean functions with a number of good combined cryptographic properties.

Plateaued Functions

A class of \( N \)-variable boolean functions \((N \text{ both odd and even})\) were introduced in [110] and termed “plateaued” functions. Before we outline the main characteristics of plateaued functions, we present the definition from [110].

**Definition 2.28.** (From [110]) Let \( f(x) \) be an \( N \)-variable boolean function with Walsh Hadamard transform vector, \( \hat{F}(\omega) \). Let \( \kappa = \{\# \omega \mid \hat{F}(\omega) \neq 0\} \). Then \( f(x) \) is a plateaued function if \( \forall \omega \in V_2^N \), the square of the elements of \( \hat{F}(\omega) \), \( (\hat{F}(\omega))^2 \in \{0, 2^{2N-t}\} \) for some even \( t \) s.t. \( \kappa = 2^t \) \((0 \leq t \leq N)\). \( f(x) \) may also be known as a plateaued function of order \( t \).

Thus, a plateaued function of order \( t \) (if \( t \neq N \)) has a three-valued Walsh Hadamard spectrum. The nonlinearity of \( N \)-variable plateaued functions is \( 2^{N-1} - 2^{N-\frac{1}{2}-1} \). The higher the order \( t \) of a plateaued function, the greater the nonlinearity of the function. For \( N \) even, plateaued functions of order \( N \) are the bent
functions. The plateaued functions of order 0 correspond to the affine functions. From Definition 2.28, the number of zeros in the Walsh Hadamard spectrum of a $t^{th}$-order plateaued function is $2^N - 2^t$. Therefore, there exist balanced and correlation immune $N$-variable plateaued functions.

It is proposed in [110] that the algebraic degree of a plateaued function $f(x)$ of order $t$ is such that $\deg(f) \leq \frac{t}{2} + 1$. Consequently, this would mean that plateaued functions do not exhibit high algebraic degree. The sum-of-square indicator of an $N$-variable plateaued function is equal to $\frac{2^{3N}}{N} = 2^{3N-t}$. As expected from the previous paragraph and the discussion of Section 2.1.3, the sum-of-square indicator will be low for large $t$.

A subset of plateaued functions is the set of partially-bent functions introduced in [7]. Unlike partially-bent functions, which always possess non-zero linear structures, plateaued functions with no non-zero linear structures exist. The reader is referred to [7] for a description of partially-bent functions. Example constructions for plateaued functions can be found in [110] and [12].

We have seen that plateaued functions may possess desirable properties such as balance, correlation immunity, high nonlinearity and low sum-of-square indicator. As with all boolean functions, the extent to which combinations of certain properties will be exhibited together are determined by the complementary or opposing nature of their relationships. Unlike bent and semi-bent boolean functions, plateaued functions may have an even or odd number of input variables.

We have discussed three special types of boolean functions of interest in this thesis. The reader should be aware that there are other special boolean functions which are not discussed, such as partially-bent functions [7].

### 2.2 S-Box Theory

In this section we now turn our discussions to the area of substitution boxes (s-boxes). The basic definitions of s-box theory are provided to support the research work performed in this thesis. Also in this section, a review of relevant cryptographic properties as applied to s-boxes, is provided.
2.2. S-Box Theory

2.2.1 S-Box Definitions and Types

A natural progression from the theory of single output boolean functions is the extension of that theory to multiple output boolean functions, collectively referred to as an s-box. The relationship between the input and output bits in terms of dimension and uniqueness gives rise to various types of s-boxes. We list below several necessary s-box definitions, together with a brief description of some s-box types of interest to this research.

An $N \times M$ substitution box (s-box) is a mapping from $N$ input bits to $M$ output bits, $S : \{0,1\}^N \mapsto \{0,1\}^M$. Thus, an s-box is simply a set of $M$ single output boolean functions combined in a fixed order. There are $2^N$ inputs and $2^M$ possible outputs for an $N \times M$ s-box. Often considered as a look-up table, an $N \times M$ s-box, $S$, is commonly represented as a matrix of size $2^N \times M$, indexed as $S[i]$ ($0 \leq i \leq 2^N - 1$), each an $M$-bit entry.

The dimensions of an s-box will have an effect on the distinctness of the output and this can influence the properties that the s-box may exhibit.

An $N \times M$ s-box with $N < M$ (a greater number of output bits than input bits) is incapable of having all possible outputs as entries in the s-box. If all output entries are distinct then the s-box is said to be injective (one-to-one).

For an $N \times M$ s-box with $N > M$ (a greater number of input bits than output bits) there must exist repeat s-box entries. If all possible outputs are present in the s-box then the s-box is said to be surjective (onto).

An $N \times M$ s-box with $N = M$ (an equal number of input and output bits) may either contain distinct entries where each input is mapped to a distinct output OR repeat s-box entries where multiple inputs may be mapped to the same output and all possible outputs are not represented in the s-box. An $N \times M$ s-box which is both injective and surjective is known as a bijective s-box. That is, each input maps to a distinct output entry and all possible outputs are present in the s-box. Bijective s-boxes may only exist when $N = M$ and are also called reversible since there must also exist a mapping from each distinct output entry to its corresponding input.

A regular $N \times M$ s-box is one which has each of its possible $2^M$ outputs appearing an equal number of times in the s-box. Thus, each of the possible output entries appears a total number of $2^{N-M}$ times in the s-box. All single output boolean functions comprising a regular s-box are balanced, as are all linear combinations of these functions. Regular $N \times M$ s-boxes are balanced s-boxes (see
Section 2.2.2 below) and may only exist when $N \geq M$.

An $N \times M$ s-box ($N \geq 2M$ and $N$ even) is said to be bent if every linear combination of its component boolean functions is a bent function.

### 2.2.2 Cryptographic Properties of S-Boxes

While many of the boolean function properties discussed in Section 2.1 have conceptual equivalences when applied to s-boxes, there are fundamental differences in the manner by which these properties are derived. As an s-box is comprised of a number of component boolean functions, it is important to observe that when considering the cryptographic properties of an s-box, it is not sufficient to consider the cryptographic properties of the component boolean functions individually. Rather, it is also necessary to consider the cryptographic properties of all the linear combinations of the component functions. This is illustrated in the following selection of relevant s-box properties.

A $N \times M$ s-box which is balanced is one whose component boolean functions and their linear combinations are all balanced. Because of this balance, there does not exist an exploitable bias in that the equally likely number of output bits over all output vector combinations ensures that an attacker is unable to trivially approximate the functions or the output.

The well-known concept of confusion due to Shannon [94] is described as a method for ensuring that in a cipher system a complex relationship exists between the ciphertext and the key material. This notion has been extrapolated to mean that a significant reliance on some form of substitution is required as a source of this confusion. The confusion in a cipher system is achieved through the use of nonlinear components. As expected, substitution boxes tend to provide the main source of nonlinearity to cryptographic cipher systems. We now define the measure of nonlinearity for an $N \times M$ s-box.

**Definition 2.29.** The nonlinearity of an $N \times M$ s-box $S$, denoted by $NL(S_{N,M})$, is defined as the minimum nonlinearity of each of its component output boolean functions and their linear combinations. Let $S = (f_1, f_2, \ldots, f_M)$ where $f_i$ ($i = 1, \ldots, M$) are $N$-variable boolean functions. Let $g_j$ be the set of linear combinations of $f_i$ ($i = 1, \ldots, M$) (which includes the functions $f_i$). Then the nonlinearity of $S$ can be expressed as follows:
2.2. S-Box Theory

\[ NL(S_{N,M}) = \min_g \{ NL(g_j) \} \ (j = 1, \ldots, 2^M - 1) \]

Clearly, as \( N \) and \( M \) increase, the task of merely computing the nonlinearity value of an \( N \times M \) s-box quickly becomes computationally infeasible. The importance of this property for the security of cipher systems becomes evident in the next section when we discuss some of the effective cryptanalytic attacks which exist.

The algebraic degree of an s-box (and similarly a boolean function) is desired to be as high as possible in order to resist a cryptanalytic attack known as low order approximation [63], [32]. The measure of s-box degree is defined below.

**Definition 2.30.** Let \( S = (f_1, f_2, \ldots, f_M) \) be an \( N \times M \) s-box where \( f_i \ (i = 1, \ldots, M) \) are \( N \)-variable boolean functions. Let \( g_j \) be the set of linear combinations of \( f_i \ (i = 1, \ldots, M) \) (which includes the functions \( f_i \)). Then the algebraic degree of \( S \), denoted by \( \text{deg}(S_{N,M}) \), is defined as

\[ \text{deg}(S_{N,M}) = \min_g \{ \text{deg}(g_j) \} \ (j = 1, \ldots, 2^M - 1) \]

A companion concept to confusion, called diffusion, was also proposed by Shannon in [94]. Therein it is described as the method by which the data redundancy in a cipher is spread throughout the entire (or large portion of the) data in an effort to reduce the probability of discovering part or all of its statistical structure. Diffusion has long been linked to the avalanche characteristics of a cipher system and, in particular, is achieved by using cipher components which exhibit good avalanche characteristics. In order to measure these characteristics for \( N \times M \) s-boxes we require the following definitions:

**Definition 2.31.** Let \( S = (f_1, f_2, \ldots, f_M) \) be an \( N \times M \) s-box where \( f_i \ (i = 1, \ldots, M) \) are \( N \)-variable boolean functions. Let \( g_j \) be the set of linear combinations of \( f_i \ (i = 1, \ldots, M) \) (which includes the functions \( f_i \)), each with autocorrelation function, \( \hat{r}_{g_j}(\alpha) \). Then the maximum absolute autocorrelation value of \( S \) is defined as:

\[ |AC(S_{N,M})|_{\text{max}} = \max_g |\hat{r}_{g_j}(\alpha)| \]
with $\alpha \in \{1, \ldots, 2^N - 1\}$ and $(j = 1, \ldots, 2^M - 1)$.

Hereinafter we shall simply use $AC(S_{N,M})_{max}$ to refer to the maximum absolute autocorrelation value of an $N\times M$ s-box, $S$.

**Definition 2.32.** Let $S = (f_1, f_2, \ldots, f_M)$ be an $N\times M$ s-box where $f_i$ ($i = 1, \ldots, M$) are $N$-variable boolean functions. Let $g_j$ be the set of linear combinations of $f_i$ ($i = 1, \ldots, M$) (which includes the functions $f_i$). Then $S$ is said to satisfy strict avalanche criterion ($SAC$) if every $g_j$ ($j = 1, \ldots, 2^M - 1$) satisfies $SAC$.

**Definition 2.33.** Let $S = (f_1, f_2, \ldots, f_M)$ be an $N\times M$ s-box where $f_i$ ($i = 1, \ldots, M$) are $N$-variable boolean functions. Let $g_j$ be the set of linear combinations of $f_i$ ($i = 1, \ldots, M$) (which includes the functions $f_i$). Then $S$ is said to satisfy propagation criteria of order $k$, $PC(k)$, if every $g_j$ ($j = 1, \ldots, 2^M - 1$) satisfies $PC(k)$.

The next two definitions outline the way in which, respectively, the correlation immunity and resilience of an s-box are determined.

**Definition 2.34.** Let $S = (f_1, f_2, \ldots, f_M)$ be an $N\times M$ s-box where $f_i$ ($i = 1, \ldots, M$) are $N$-variable boolean functions. Let $g_j$ be the set of linear combinations of $f_i$ ($i = 1, \ldots, M$) (which includes the functions $f_i$). Then $S$ is a CI($t$) s-box if all $g_j$ ($j = 1, \ldots, 2^M - 1$) are CI($t$) boolean functions.

Similarly,

**Definition 2.35.** Let $S = (f_1, f_2, \ldots, f_M)$ be an $N\times M$ s-box where $f_i$ ($i = 1, \ldots, M$) are $N$-variable boolean functions. Let $g_j$ be the set of linear combinations of $f_i$ ($i = 1, \ldots, M$) (which includes the functions $f_i$). Then $S$ is a $t$-resilient s-box if all $g_j$ ($j = 1, \ldots, 2^M - 1$) are $t$-resilient boolean functions.

### 2.3 Some Common Cryptanalytic Attacks on Cipher Systems

In this chapter we have outlined a number of cryptographic properties of boolean functions and s-boxes relevant to this research. The manner in which they contribute to the security of single and multiple output functions, and thus cipher systems as a whole, is best illustrated by discussing some of the existing cryptanalytic attacks and how those properties provide resistance to them.

We now briefly summarize three common methods for attacking cipher systems.
2.3.1 Differential Cryptanalysis

The technique of differential cryptanalysis is due to Biham and Shamir [4] and is essentially applied to block ciphers in a chosen-plaintext attack. A basic differential attack on a block cipher system involves the analysis of trends between plaintext input differences and corresponding output differences in the ciphertext. In general, a differential attack seeks to exploit these trends in order to gain knowledge of information such as key bits which may reduce the computational complexity of breaking the cipher.

For a block cipher with block length $B$ bits, let $X = X_1 \ X_2 \ X_3 \ldots \ X_B$ and $Y = Y_1 \ Y_2 \ Y_3 \ldots \ Y_B$ represent a plaintext and ciphertext block respectively. If $X^i$ and $X^j$ are two $B$-bit blocks of plaintext with $Y^i$ and $Y^j$ their corresponding output blocks, then the input and output differences are $\Delta_X = X^i \oplus X^j$ and $\Delta_Y = Y^i \oplus Y^j$ respectively. A corresponding pair of input and output differences is called a differential. Attached to each differential is a probability that the output difference will occur a certain number of times, given the input difference, i.e. $Pr(\Delta_Y|\Delta_X)$. The lower the differential probability, the less likely the output difference is to occur for a particular input difference. This is desirable as we wish to minimize the correlation between input and output differences which in turn make accurate predictions of intermediate bits during the encryption process more difficult. A series of differentials for consecutive rounds in the cipher which satisfy $\Delta_X^k = \Delta_X^{k+1}$ for rounds 1 to $l$ is called an $l$-round differential characteristic. Differential characteristics are used to determine the overall differential probability of a cipher. Thus, a differential characteristic probability may be measured by calculating the product of the probabilities of each individual round differential, assuming of course that they are independent of each other.

Differential probabilities are influenced by the cipher components within the rounds of the cipher. Substitution boxes form a key component of block ciphers with respect to their security as they provide essential and often sole source of nonlinearity to the system. An s-box differential is the two values representing the difference between two input values and the difference between their corresponding output values. For an $N \times M$ s-box, there are $2^{2N-1} - 2^{N-1}$ possible distinct input pairs, producing input differences from a possible $2^N$ distinct values. Tabulating the frequency of occurrence of all the resultant output differences, of which $2^M$ distinct values are possible, form the basis of the difference distribution table of the s-box. Thus, the difference distribution table is a $2^N \times 2^M$ matrix containing...
the frequency of occurrences of all possible output differences given each possible input difference. The largest value in the difference distribution table of an s-box is commonly written as $\delta$, and referred to as differential uniformity [91].

The values in each row of the difference distribution table must sum to $2^N$ since an input difference exists for the pairing of every possible distinct input to the s-box. Therefore, a flat difference distribution table, which is one where the frequency values are almost uniform, implies that the magnitude of the frequencies are small. An s-box whose difference distribution table is flat provides little or no information about output differences which may be exploited to reveal intermediate bits of the cipher. Large frequency values in the difference distribution table can be used to form a differential characteristic with high probability.

In a typical cipher system, several rounds of processing occur with multiple s-box look-ups. By combining s-box differentials, a differential characteristic probability for the cipher system can be determined. In order for a cipher to successfully resist differential cryptanalysis, the differential characteristic probability should be small. Ciphers which contain a greater number of rounds are likely to be better able to achieve a low probability differential characteristic. The magnitude of s-box differentials will also affect the differential characteristic probability of the overall cipher. The absence of any high values in the difference distribution table of the s-box result in small s-box differential probabilities and thus produces a differential characteristic with low probability.

Let $D_S$ be a $2^N \times 2^M$ matrix representing the difference distribution table of an $N \times M$ s-box, $S$. Let $A_S$ be a $2^N \times 2^M$ matrix representing the autocorrelation matrix of $S$. It has been shown in [108] that, for an $N \times M$ s-box with $N \geq M$, the relationship between its difference distribution table and autocorrelation matrix is given by the matrix product $D_S \hat{L}$ where $\hat{L}$ is the polarity form of the linear matrix (also known as the Sylvester-Hadamard matrix [60], Chapter 2). A lower bound on the differential uniformity, $\delta$, involving the maximum absolute value in the autocorrelation matrix of an s-box is given in [108] as follows:

$$\delta \geq 2^{N-M} + 2^{-M} AC(S_{N,M})_{\max}$$

where $AC(S_{N,M})_{\max}$ is as defined in Definition 2.31. Noting that $\delta$ takes a value in the range $[2^{N-M}, 2^N]$, an $AC(S_{N,M})_{\max}$ value of 0, exhibited by bent s-boxes, results in the minimum $\delta$ value possible. Additionally, the presence of non-linear structures in the s-box will consequently cause $AC(S_{N,M})_{\max}$ to take the value
2^N. A further observation made in [108] is that a small \( \delta \) implies a small value for \( AC(S_{N,M})_{max} \). Hence, minimizing the overall autocorrelation of s-boxes (in terms of their maximum absolute autocorrelation value) helps to provide resistance to differential cryptanalysis through the minimization of their differential uniformity and in turn reducing the differential characteristic probability of the cipher.

In [108], two upper bounds on the nonlinearity of an \( N \times M \) s-box are provided which relate it to the enumeration of non-zero entries in the difference distribution table of the s-box and depend also on \( N \) and \( M \). The reader is referred to Theorems 3 and 4 of [108] for details. In essence, an increase in the number of non-zero entries in the table corresponds to an s-box with potentially higher nonlinearity. Conversely, a highly nonlinear s-box forces a minimum number of non-zero entries in its difference distribution table, thus reducing its susceptibility to differential cryptanalysis.

The Data Encryption Standard (DES) [74] was shown to be breakable by differential cryptanalysis [4] [5] in the early 1990s. The susceptibility of DES was primarily due to the fact that the difference distribution tables of the DES s-boxes exhibit clear non-uniformity, whilst resistance against differential cryptanalysis is characterized by a highly uniform difference distribution table, as discussed earlier.

In summary, the resistance of cipher systems to differential cryptanalysis is enhanced by the incorporation of s-boxes with strengthening properties such as high nonlinearity and low autocorrelation which each contribute to improving this resistance.

### 2.3.2 Linear Cryptanalysis

Linear cryptanalysis, introduced by Matsui [57] in 1993, is a form of known plaintext attack which attempts to approximate the relationship between plaintext, ciphertext and key bits by forming a linear expression and evaluating the probability of that expression accurately depicting the relationship. In this way, the goal of a linear cryptanalytic attack is to reveal bits of the key.

For a block cipher with block length \( B \) bits, let \( X = X_1 \ X_2 \ X_3 \ldots X_B \) and \( Y = Y_1 \ Y_2 \ Y_3 \ldots Y_B \) represent plaintext and ciphertext blocks respectively. Linear cryptanalysis is concerned with finding a linear expression for some combination
of input and output bits where

\[
\bigoplus_{i=1}^{B} \psi_i X_i = \bigoplus_{j=1}^{B} \tau_j Y_j
\]  

(2.1)

with \( \psi_i, \tau_j \in \{0,1\} \). The best linear approximation is the expression with the highest probability of being valid, and the best affine approximation is the expression with the lowest probability of being valid. Let \( P = Pr(X = Y) \) be the probability associated with the above expression. If \( P \approx \frac{1}{2} \), then this indicates that the cipher is highly resistant to linear or affine approximation. Thus, the probability bias is given by \( | P - \frac{1}{2} | \), the variation away from the expected probability for a random process.

Any linear expression which seeks to relate the plaintext, ciphertext and key bits of a cipher must include for consideration, the structure of the cipher and the components, including any s-boxes utilized in the rounds. To find a linear approximation to an \( N \times M \) s-box, the linear relationships between inputs and outputs of the s-box may be calculated for all pairs of inputs and outputs. This is expressed in a \( 2^N \times 2^M \) matrix referred to as the linear approximation table.

Each entry in the linear approximation table, \( L_{X',Y'} \), of an s-box can be calculated as

\[
L_{X',Y'} = 2^{N-1} - hd(X',Y')
\]

This value also provides the signed probability bias, \( \frac{L_{X',Y'}}{2^N} = Pr(X' = Y') - \frac{1}{2} \), a real value in the range \([-\frac{1}{2}, +\frac{1}{2}]\). A probability bias of 0 indicates that no linear approximation is possible, while a bias approaching \( \pm \frac{1}{2} \) indicates that the s-box can easily be approximated by a linear or affine function. Thus, the best linear approximation to an \( N \times M \) s-box will be the linear expression of the form in equation 2.1 whose input and output bits, \( X' \) and \( Y' \) respectively, correspond to the entry with the largest magnitude in the linear approximation table of the s-box.

In general, applying linear cryptanalysis to a block cipher system involves finding a linear approximation with a large signed probability at each stage of the cipher, typically for the rounds. The ability to combine probabilities of linear equations best approximating different stages of the encryption process is reliant upon the assumption of independence of the linear approximations at each stage.
Matsui makes use of the Piling-up Lemma (Lemma 3 of [57]) to define the overall probability of a linear expression which necessarily combines the probabilities of multiple linear approximations all holding under this assumption. The overall linear approximation of the cipher is arrived at by linking multiple linear expressions together whilst achieving any length reduction by cancelling pairs of common terms. The application of the Piling-up Lemma provides the probability for the cipher’s overall linear approximation. The higher the probability calculated from the Piling-up Lemma, the more likely the approximation will successfully retrieve relevant bits of the key given sufficient plaintext-ciphertext pairs. Further, the greater the magnitude of the bias exhibited by the individual linear expressions at each stage of the process, the higher the overall probability of approximating the cipher linearly. For linear approximations to a component s-box, bias values in its linear approximation table which are disproportionally high will result in a more successful cryptanalytic attack on the cipher system.

The values in the linear approximation table of an $N \times M$ s-box are closely related to the entries in the Walsh Hadamard transform matrix of all linear combinations of component boolean functions of the s-box. For an s-box, the probability that some linear expression of inputs is equal to some linear expression of the outputs, $Pr(X' = Y')$, is measured by the number of instances in which the input is equal to the output (for all possible inputs), out of $2^N$. This is equivalent to $\frac{2^N - \text{hd}(X', Y')}{2^N}$ where the hamming distance is measured in terms of the \{0,1\} (truth table) forms of the input and output. The probability bias is the amount by which $Pr(X' = Y')$ deviates from $\frac{1}{2}$. The values in the linear approximation table of an s-box represent the bias (ie $2^N \times$ probability bias).

The Walsh Hadamard transform matrix of an s-box computes the relationship between the input, $X'$, of the s-box (a linear function) and the output, $Y'$, of the s-box. This relationship is also measured by the distance between expressions, but in terms of the \{1,-1\} (polarity) forms of the input and output. Therefore, the Walsh Hadamard transform matrix entries give the values of $2^N - 2\text{hd}(X', Y')$ for each linear function $X'$ and each output $Y'$. A highly nonlinear output $Y'$ reduces the magnitude of the matrix entries. The bias, as discussed above, can be directly obtained from the Walsh Hadamard transform values by:

\[
\text{bias} = L_{X', Y'} = \frac{\hat{F}(\omega)}{2}
\]
where $\hat{F}(\omega)$ is defined in Definition 2.13. This in turn is directly related to the nonlinearity of the s-box by:

$$NL(S_{N,M}) = 2^{N-1} - |L_{X',Y'}|_{max}$$

where $X' \neq 0$, $Y' \neq 0$, and $|L_{X',Y'}|_{max}$ represents the maximum absolute value in the linear approximation table.

For large $N$, $M$ the computation of the entire linear approximation table for an $N \times M$ s-box is infeasible. Nevertheless, the incorporation of highly nonlinear s-boxes into cipher systems is desirable in order for the cipher to be resistant to linear cryptanalytic attacks.

In 1993, [57] showed that DES was also breakable by linear cryptanalysis. This was due to the existence of high magnitude values in the linear approximation tables of the DES s-boxes. As mentioned earlier, resistance against linear cryptanalysis requires low magnitude values in the linear approximation table, which is obtained through the use of highly nonlinear s-boxes.

We conclude that high nonlinearity is an important property for the security of cipher systems and components.

### 2.3.3 Correlation Attacks

The concept of a correlation attack was first proposed by Siegenthaler [95] in 1984. This has since given rise to a number of specific variants of the attack in [95] such as Fast Correlation Attacks [61], Divide-and-Conquer Attacks [96] [23], Decimation Attacks [30], still all collectively known as correlation attacks. Typically, modern stream ciphers use combination keystream generators such as those which comprise multiple linear feedback shift registers (LFSRs) linked together with a nonlinear combining function. Correlation attacks work on analyzing the correlation between a sequence of output bits from one or more LFSRs and the keystream. The uncorrelatedness between the resulting keystream and some fixed subset of $m$ input variables to the combining function from the individual LFSRs determines the order of correlation immunity, $m$.

In a typical stream cipher, each of $N$ individual LFSRs has an initial state, which is often the target of the cryptanalytic attack. The basic correlation attack seeks to determine the most significant correlation between the output of the target LFSRs (a subset of the input to the combining function), using every
possible initial state, and the output of the combining function. A combining function which is $CI(m)$ will exhibit statistical independence from up to and including $m$ target LFSRs. The longer the length of the LFSRs and/or the more LFSRs are targeted, the more inefficient and ineffective this attack is.

The fast correlation attack proposed by Meier and Staffelbach [61] uses a series of parity check equations which are determined from the feedback polynomial of the LFSR. A probability that all of the parity check equations hold for each bit in the keystream is calculated. The bit positions with the highest probabilities are then used to form a proposed candidate initial state. Small changes are made to this candidate until the state with perfect correlation is found. Greater efficiency is achieved as the parity check equations are able to be computed much quicker than the exhaustive search process of [95]. Various enhancements to the fast correlation attack have been made, see for example, [40] and [41]. The Decimation Attack [30], mentioned above, is a further variation which involves approximating a long LFSR with several shorter LFSRs.

The incorporation of an appropriate combining boolean function $f$ can increase a stream cipher’s resistance to correlation attacks. In particular, if $f$ is highly nonlinear, fast correlation attacks become infeasible as a greater distance between $f$ and the closest affine function prevents a good linear approximation being made by the parity check equations. In addition, if $f$ is correlation immune to some order $m$, the output of $f$ is not correlated to any fixed subset of $m$ input variables, thus increasing the resistance of $f$ to correlation attacks, as discussed above. Two other necessary properties to avoid cryptographic weaknesses in the combining function are balance and high algebraic degree, desired to prevent output bias and maintain high algebraic complexity. Note that a trade-off between the nonlinearity and correlation immunity of $f$ must be considered, and is discussed in Section 2.1.3. A further trade-off exists between the number of inputs to $f$, order of correlation immunity and algebraic degree of $f$, as shown in Theorem 2.5.

### 2.4 Summary

In this chapter we have defined the relevant supporting theory of both boolean functions and substitution boxes. In particular, we have provided numerous long established definitions and theorems for various aspects of the theory. The nec-
ecessary cryptographic properties which are used to analyze the strength of single and multiple output functions have also been defined and discussed, as have the inter-relations between pairs of selected properties. Finally, we have presented a brief summary of major cryptanalytic attacks against boolean functions, substitution boxes and cipher systems. From the discussions in this chapter, it is clear that cryptographic cipher systems are most vulnerable to attack if exploitable weaknesses in their components exist. For this reason, it is important to be able to ensure that any boolean functions and s-boxes incorporated into a cipher system exhibit the appropriate combination and measures of robust cryptographic properties necessary for that particular type of cipher and its use.

Methods are needed to obtain strong boolean functions and s-boxes satisfying desirable cryptographic properties. Such methods may also be utilized as a tool for the analysis of these cipher components. This is the focus of the following chapters.
The focus of this PhD research has been on the analysis, investigation and optimization of boolean functions and substitution boxes. The focus of optimization efforts has been on improving the cryptographic properties of these cipher components. As the size of an input space increases, it quickly becomes infeasible to exhaustively search the space in order to analyze the properties exhibited by functions within the space. Thus to discover knowledge about functions, particularly within a large search space, it is necessary to employ techniques to direct investigations to certain parts of the space which contain functions of interest, typically those which exhibit one or more desirable cryptographic properties. Desirable properties are generally those which contribute to the security of the function.

The two main techniques which have been used for this purpose by researchers in the field are:

1. Heuristic techniques; and
2. Algebraic constructions.

Heuristic techniques are driven by a directed search algorithm typically searching in a localized area from a specified starting point. Their use is more frequent for searching in large spaces in order to find a large number of solutions which are satisfactory, but generally not optimal. For this reason, heuristic techniques are often applied to difficult combinatorial problems. Well known heuristic techniques include Simulated Annealing [43], Tabu Search [31], Genetic Algorithms
Algebraic constructions rely on proven mathematical relationships holding for a generalized construction of functions. Whilst algebraic constructions have been shown to generally produce functions with the most optimum combinations of properties, they are not typically designed to produce a great number of such functions. Further, the existence of inherent weaknesses in functions produced by algebraic construction is a valid concern. In contrast, the vast amount of experimentation so far performed using heuristic techniques has shown that, for large input spaces, these techniques are generally unable to generate optimal functions. This is due to the nature of the technique as simply being a way to non-deterministically search through a search space in a directed fashion. Thus, as the number of input variables increases by one, the number of functions in the space increases by a factor of $2^{2^n}$ and the probability of discovering optimal functions decreases. However, because heuristic techniques involve directed search methods, they have been shown to produce consistent results in finding functions with “good” properties, and unlike algebraic constructions, are able to produce a large number of such functions. For this reason, the approach taken in this research has been primarily focussed on the application of heuristic techniques.

This chapter is devoted to describing the existing heuristic techniques and methods which have been applied in this research work. The first of two sections in this chapter is essentially an overview of the particular existing heuristic techniques which have been utilized in this thesis for s-box optimization. The chapter concludes with a summary of the relevant existing heuristic techniques.

### 3.1 Overview of Existing Heuristic Techniques Used

For many years, Genetic Algorithms and Hill Climbing techniques have been used as heuristic optimization techniques for non-cryptographic applications. For example, a form of hill climbing was used as early as 1961 by Minsky [70], applied to developing artificial intelligence systems. In 1973, Holland [37] introduced the concept of Genetic Algorithms, based upon implementing a version of Darwin’s “survival of the fittest” principle [21] to a study of cellular automata.

We now discuss their cryptographic application, specifically in generating
3.1. Overview of Existing Heuristic Techniques Used

strong components for use in cipher systems to enhance their security.

3.1.1 Hill Climbing

Since the late 1990s, hill climbing techniques have been demonstrated to be effective in research of a cryptographic nature. A significant amount of important research has been performed in using hill climbing to optimize the cryptographic properties of both boolean functions [66], [68] and s-boxes [64]. Hill climbing has also been used in conjunction with other heuristic techniques such as Simulated Annealing [16], [17] and Genetic Algorithms [65], [67].

The basic hill climbing technique involves searching, at each iteration, for elements of a function to modify which will result in an improvement in the results already obtained. At the end of the process, it is expected that the final output will represent the best solution obtainable.

For cryptographic applications used in this research, hill climbing is referred to as being the process whereby one or more distinct elements in the truth table of a function are complemented in order to make iterative improvements to the cryptographic properties or fitness of the function.

The fitness of a function is the measure of a particular cryptographic property or properties exhibited by the function. In [68], the authors categorize the fitness function into either weak or strong acceptance. A weak acceptance condition will accept an incremental change in the truth table even if such a change produces no increase in the fitness of the new function, provided that there is no decrease in the fitness. A strong acceptance condition, on the other hand, will only accept an incremental change in the truth table when such a change produces an increase in the fitness of the new function. Thus, the only time an increase in the fitness is forced is when a strong acceptance condition is imposed.

In addition to relying on this measure as a criterion for deciding whether to accept or reject functions to be input into the next iteration of the process, hill climbing requires the formation of improvement sets.

Improvement sets are defined according to the fitness function which is utilized in the hill climbing process. If we consider the use of nonlinearity as the fitness measure, the corresponding improvement sets represent those positions in the Walsh Hadamard transform vector of the boolean function which take on values which may affect the fitness of the function as a direct result of the complementing of elements in the function’s truth table. It is well known that one and two
bit changes in the truth table of a boolean function results in a change in the Walsh Hadamard transform of the function $\in \{2,-2\}$ and $\{0,4,-4\}$ respectively [64]. Thus, for two bit changes in the truth table of each boolean function, the six improvement sets which are calculated in the hill climbing process are [64]:

$I_1 = \{\phi : \text{WHT}(\phi) = \text{WHT}_{\max}\}$

$I_2 = \{\phi : \text{WHT}(\phi) = -\text{WHT}_{\max}\}$

$I_3 = \{\phi : \text{WHT}(\phi) = \text{WHT}_{\max} - 2\}$

$I_4 = \{\phi : \text{WHT}(\phi) = -(\text{WHT}_{\max} - 2)\}$

$I_5 = \{\phi : \text{WHT}(\phi) = \text{WHT}_{\max} - 4\}$

$I_6 = \{\phi : \text{WHT}(\phi) = -(\text{WHT}_{\max} - 4)\}$

where $\text{WHT}_{\max}$ is as in Definition 2.17.

These represent the only values which may affect the nonlinearity of a boolean function after a two bit change in the truth table of a boolean function has taken place. Let $\mathcal{I}(f) = \bigcup_{i=1}^{6} I_i$ represent the union of improvement sets for boolean function, $f$.

The general algorithm for hill climbing boolean functions by two bit changes is described in Algorithm 3.1 and is referred to as the Hill Climbing Method.

A slight modification to this algorithm may be made to accommodate the hill climbing of s-boxes by randomly generating a starting s-box to hill-climb. It is then necessary to calculate the fitness of each of the linear combinations of the component boolean functions of the s-box, and iterate step 3 of Algorithm 3.1 for the hill-climbed s-box. The final s-box produced in the output will represent the s-box displaying the best fitness.

The application of hill climbing to boolean function property optimization [66], [68] is summarized in the Previously Reported Results section below.

**Experimental Rationale**

The hill climbing process requires a fitness function to be defined in order to target one or more specific cryptographic properties to be optimized. The fitness function is the measure of the target property and the decision to accept or reject progressive output functions is based on this measure. Much of the work which has been done in optimizing boolean function properties using this heuristic technique have utilized a fitness function defined to assess the nonlinearity measure.

For the Hill Climbing Method described above, the key step in the process is
3.1. Overview of Existing Heuristic Techniques Used

Algorithm 3.1: Hill Climbing Method

1. Define \( \text{fit}(f) \) as the fitness function of a boolean function, \( f \).
2. Generate a random \( N \)-variable boolean function, \( f(x), x = 0,\ldots,2^N - 1 \).
3. Iterate:
   - (a) Form improvement sets \( \mathcal{I}(f) \) and compute \( \text{fit}(f) \).
   - (b) Select two bit positions, \( i \) and \( j \), where \( f(i) \neq f(j) \).
   - (c) Derive candidate function \( g(x) = f(x) \), where \( x = 0,\ldots,i-1,i+1,\ldots,j-1,j+1,\ldots,2^N - 1 \); and \( g(i) = f(i) \oplus 1, g(j) = f(j) \oplus 1 \).
   - (d) Form improvement sets \( \mathcal{I}(g) \) and compute \( \text{fit}(g) \). If \( \forall \phi \in \mathcal{I}(g), |WHT(\phi)| < WHT_{\text{max}} \), then let \( f = g \).
   - (e) If loop reaches limit of iterations with no further improvements, exit.
   - (f) Output final boolean function, \( g \), representing best achievable fitness.
4. Repeat 2. and 3. as required, to hill climb desired number of boolean functions.

The swapping of two distinct elements in the truth table of the function so that iterative improvements may be made to the target property. Nonlinearity is the most natural fitness measure to adopt for this process because of the direct effect that such a swap has on the Walsh Hadamard transform of a boolean function and thus the corresponding nonlinearity value. In particular, a one bit change in the truth table of a boolean function produces a change to the Walsh Hadamard transform of the function \( \in \{ \pm 2 \} \) forcing the nonlinearity measure to change by \( \{ \pm 1 \} \) (see Lemma 2.7 and Corollary 2.1). Similarly, a change to the Walsh Hadamard transform of a boolean function \( \in \{ 0,\pm 4 \} \) as a result of a two bit change in the function’s truth table forces a change in the nonlinearity measure \( \in \{ 0,\pm 2 \} \) (see Lemma 2.8 and Corollary 2.2).

The focus of experiments using hill climbing in papers such as [66] and [68] has been in finding boolean functions which exhibit the best achievable measure in one or more target cryptographic properties.
Previously Reported Results

In order to summarize previous work which has been done on boolean function property optimization by applying hill climbing, we provide results from relevant research papers illustrating the extent to which these techniques have been successful.

In [66], the results of applying hill climbing were reported which involved both a one and two-bit change to the truth table of candidate boolean functions. Experiments concentrated on comparisons between the effectiveness of random generation in achieving boolean functions with high nonlinearity and the effectiveness of hill climbing randomly generated boolean functions to improve their nonlinearity. The random case was used in [66] as a baseline for comparison purposes to determine the degree of success of the application of hill climbing to this problem.

Graphical results are shown in [66] which provides a probability -v- nonlinearity comparison of their Hill Climbing Method with that of random generation of balanced and unbalanced boolean functions for $N = 8$ and 12. It is clear from these graphs that the frequency of generating higher nonlinearity boolean functions is significantly greater than that which is able to be achieved by pure random function generation. Also, the Hill Climbing Method is consistently able to achieve higher nonlinearities than random generation. In these particular experiments, the highest nonlinearity reported to be obtained by the Hill Climbing Method was 112 for $N = 8$ and 1958 for $N = 12$.

The paper of [68] included a study of the applicability of hill climbing to improving the nonlinearity and autocorrelation of boolean functions. In particular, improvement conditions were proposed as either being weak or strong, in relation to the retention of a change in the truth table during the hill climbing process. These weak and strong hill climbing improvement conditions are formally set out in Figure 1 of [68] for nonlinearity (measured in terms of the Walsh Hadamard transform) and autocorrelation (in terms of the maximum absolute autocorrelation value).

The results of 8-variable boolean function experiments in [68] are displayed graphically. These all compare the weak and strong options for nonlinearity and autocorrelation ($AC_{max}$) in turn when using hill climbing against those of random generation. It was shown from the graphs that at all times the curve representing random generation produced the worst results for nonlinearity and
autocorrelation. When each of nonlinearity and autocorrelation were trialled
with their strong options, and then options varied for the converse property, the
general trend was that highest target property values were equally likely to be
achieved whether using the weak option of the converse property or ignoring the
converse property altogether. When each property was trialled with the strong
option of the converse property, allowing the options for the original property to
vary, the best results were obtained by the strong option for that property.

The frequency of highest nonlinear boolean functions was achieved by the
strong nonlinearity option and either the weak or no autocorrelation option. Sim-
ilarly, the frequency of boolean functions with low $AC_{max}$ was achieved by the
strong autocorrelation option and the weak or no nonlinearity option.

Method Applicability

The past research discussed above highlighted the effectiveness of the application
of hill climbing to boolean function generation in order to improve nonlinear-
ity and autocorrelation (in terms of maximum absolute autocorrelation value).
This represented a significant improvement over the achievable results of random
generation alone.

Later boolean function work, as described in this thesis, has now surpassed
the achievement of hill climbing for high nonlinearity and low autocorrelation.
Hill climbing still remains a very useful technique for the generation of boolean
functions with good cryptographic properties. However, the property measures
achievable by hill climbing are reduced as $N$ increases.

The Hill Climbing Method (s-box variation) has been used in this research
to generate $N x M$ regular s-boxes ($N > M$) possessing good nonlinearity and
autocorrelation measures. In addition, an alternative approach to the efficient
generation of MARS-like s-boxes (9x32) with good cryptographic properties was
achieved by applying the Hill Climbing Method. We discuss, in depth, these and
other applications of the Hill Climbing Method in Chapters 5 and 6 of this thesis.

3.1.2 Genetic Algorithms

Genetic Algorithms were first utilized for cryptographic purposes in the early
1990’s as tools for the cryptanalysis of classical ciphers [58], [97]. More recently,
Genetic Algorithms have been applied with similar objectives as the Hill Climbing
Method in the generation of boolean functions in an effort to make improvements
to a number of important cryptographic properties known to provide more security [65], [67].

The concept behind Genetic Algorithms involves the mechanism of evolution known as natural selection. During the process of natural selection, a population of parents interbreed to produce children. Some mutation may occur, followed by a selection process where only the fittest individuals survive to become the next generation. Genetic Algorithms adopt this principle idea which is fundamental to the theory of evolution.

We begin by defining some terminology relating to natural selection:

- **parent pool** - contains the current set of candidate solutions.
- **parents** - the pair of individuals in the parent pool chosen for breeding.
- **children** - offspring resulting from the breeding of two parents.
- **breed** - the process whereby two parents are combined or mated to produce a child.
- **fitness** - the measure taken in order to ascertain which individuals will survive to the next generation.

The application of Genetic Algorithms to the research performed in this thesis defines the breeding population as the parent boolean functions or substitution boxes, which breed to produce children that possess some of the characteristics of both parents. If mutation is applied, it is introduced into the whole population in the form of resetting. Resetting is a procedure used to introduce some further randomness into the algorithm. Like the Hill Climbing Method, Genetic Algorithms make use of a fitness function to ascertain the measure of the fitness of individuals in the population pool. Only the best solutions are selected to continue into the next generation.

Let $T$ be the number of initial starting functions. A description of a basic Genetic Algorithm for improving the cryptographic properties of boolean functions is given in Algorithm 3.2.

The steps in Algorithm 3.2 outline the general procedure for the application of a basic Genetic Algorithm to randomly generated boolean functions. There are a number of variations of this algorithm which could be implemented depending on the types of results targeted. Some variations to the basic Genetic Algorithm described in Algorithm 3.2 are discussed in Chapter 5 of this thesis. Experiments conducted using these variations, and their effects on the overall results are also discussed. The reader will note that in order to apply the basic Genetic Algorithm
Algorithm 3.2: Genetic Algorithm

1. Define $fit(f)$ as the fitness function of a boolean function, $f$.

2. Define the breeding function of two $N$-variable boolean functions, $f$ and $g$, as $breed(f, g)$.

3. Generate a set of $T$ random $N$-variable boolean functions, representing the parent pool, $P_i$ where $i = 1, ..., T$.

4. Iterate:
   (a) Generate a set of \( \frac{T(T-1)}{2} \) boolean functions (children), $C_i = breed(P_j, P_k)$, where $j = 1, ..., T$, $k = 1, ..., T$, $j \neq k$.
   (b) Form sorted, combined set $S = P \cup C$, where $S = \{S_1, S_2, ..., S_{T+\frac{T(T-1)}{2}}\}$ and $fit(S_l) \geq fit(S_{l+1})$, $l = 1, 2, ..., T + \frac{T(T-1)}{2} - 1$.
   (c) Keep best $T$ functions, by replacing new $P_i = S_i$ where $i = 1, ..., T$.
   (d) Reset: optional.
   (e) Iterate until specified stopping criteria is met.

5. Repeat 3. and 4. as required, to compute genetic algorithm for a desired number of initial pools.

To s-boxes it is simply a matter of defining the starting pool to consist of a set of $T$ s-boxes, and by defining the breeding function to operate on s-boxes instead of boolean functions.

We illustrate the general process of breeding by describing below two common schemes for breeding functions in a population pool.

**Roulette Wheel** [38]: This scheme arises from the idea that the parents in the population pool occupy a particular percentage angle on a roulette wheel, the size of which is determined proportionately by their fitness measure. Thus, more fitter individuals occupy greater angles on the wheel and have a higher chance of selection with the spinning of the wheel.

**Crossover**: This breeding scheme is based on the genetic mechanism of crossover which occurs in sexual reproduction. In this natural process, genetic variation results from the breaking and recombination of linked genes in homolo-
gous chromosomes, thus producing offspring with combined attributes of two parents. Function breeding schemes based on crossover extend this idea by choosing a random position in the two parent functions at which the crossover of elements will begin and subsequently interchanging the elements in the parent functions from this point. Thus, the resulting offspring will take the elements of the first parent up to and including the element at the crossover point and the elements of the second parent for the remaining positions [22].

Resetting

An optional variation to a Genetic Algorithm is the use of resetting. Resetting is a procedure used to provide the computational process with an additional element of randomness and is often introduced to alter the direction of convergence. Changing the direction of convergence may enable a Genetic Algorithm process to deviate away from a point of local maxima which occurs because of the number of local peaks of fitness which exist across the entire search space. There are a number of ways in which a further random component may be introduced into a Genetic Algorithm. For example, at a point when the change in fitness becomes increasingly small, by replacing $r$ parents ($1 < r \leq T$) in a random fashion, it is expected that the process will have a greater probability of directing the search to more distant areas of the search space.

We summarize, in the Previously Reported Results section below, the results of previous work where Genetic Algorithms have been applied for the purposes of optimizing boolean function properties [65], [67].

Experimental Rationale

Like hill climbing techniques, boolean function property optimization using Genetic Algorithms requires the algorithms to define a fitness function based on the target cryptographic property or properties which the method is seeking to optimize. Experiments with Genetic Algorithms to date have largely utilized a fitness function measuring the nonlinearity property.

A basic Genetic Algorithm for improving the cryptographic properties of boolean functions and s-boxes is described above. The main component of a Genetic Algorithm responsible for adapting the functions in the pool to produce fitter individuals is the breeding function. Genetic Algorithms are driven by the expectation that genetic variation (as a direct result of the breeding scheme used)
will produce functions in the population pool with better fitness throughout the generations. Thus, the breeding scheme is vital to the overall goal of producing functions with the most optimal fitness achievable by Genetic Algorithms.

The focus of experiments using Genetic Algorithms in papers such as [65] and [67] has been in finding boolean functions which exhibit the best achievable measure in one or more target cryptographic properties. The specifics of these past experiments are discussed below.

**Previously Reported Results**

We now outline the results of experiments from selected research papers based on boolean function property optimization using Genetic Algorithms and the nature of those experiments.

Millan et al. [65] investigated the effects of applying a Genetic Algorithm to single output unbalanced boolean functions in order to improve their nonlinearity property. Experiments were conducted on 8 to 16-variable unbalanced boolean functions for a sample size of 1000 and 10000 functions and compared the nonlinearity achievable by the Genetic Algorithm with the random case as well as the same application using hill climbing. It was reported that, in general, for a sample size of 1000 boolean functions, the Hill Climbing Method either equalled or slightly outperformed the Genetic Algorithm. Both the Genetic Algorithm and the Hill Climbing Method were noticeably superior to that of random generation. For a sample size of 10000 functions, however, each of the Genetic Algorithm and Hill Climbing Method produced slightly better results for particular values of $N$ than the other, seemingly indicating their equal effectiveness for this larger sample size. Again, their results were far superior to random generation alone.

The application of a Genetic Algorithm to single output balanced boolean functions to improve nonlinearity was investigated in [67] where an optional resetting feature was also incorporated into the algorithm. The results of experiments in [67] for typical pool sizes of 30 included best achievable nonlinearity values for 8 to 12-variable boolean functions comparing the Genetic Algorithm with and without resetting. These results indicated that a Genetic Algorithm with resetting more often produced a higher achievable nonlinearity value than a Genetic Algorithm without resetting.
Method Applicability

Genetic Algorithms have been shown to be suitable for generating balanced and unbalanced boolean functions with improved nonlinearity values for $N \leq 16$. The increased effectiveness of incorporating a resetting feature into the algorithm for slightly improved results, has been demonstrated. Although the previous research work discussed above provided no computational times for the algorithm, Genetic Algorithms are known to be computationally intensive for function property optimization, particularly for large $N$ and for multiple outputs, compared to say the Hill Climbing Method.

In this research we have applied the Genetic Algorithm (s-box variation) to random generations of regular $N \times M$ s-boxes ($N > M$) in order to improve the nonlinearity property of the s-boxes. We discuss the results of our experiments in Chapter 5, together with the efficiency of the algorithm.

3.1.3 Combined Genetic Algorithm and Hill Climbing

The application of combining a Genetic Algorithm with the Hill Climbing Method for cryptographic research was first performed in [67]. The authors applied a combined method to random generations of single output balanced boolean functions in order to make improvements to certain cryptographic properties. It was experimentally demonstrated in [67] that single output balanced boolean functions exhibiting better target cryptographic properties could be obtained by using a combination of both Genetic Algorithm and hill climbing than obtained by either technique used in isolation.

The combining of the two techniques in [67] involved each of the $\frac{T(T-1)}{2}$ children bred from the functions in the parent pool being hill climbed. This was performed prior to calculating their fitness and subsequent sorting of parents and children together in order of best to worst fitness.

We outline the general algorithm for this combined Genetic Algorithm and hill climbing procedure applied to boolean functions, in Algorithm 3.3.

Previous work on the application of the combined heuristic techniques of a Genetic Algorithm and the Hill Climbing Method ([65], [67]) are summarized in the Previously Reported Results section below.
Algorithm 3.3: Combined Genetic Algorithm/Hill Climbing

1. Define $fit(f)$ as the fitness function of a boolean function, $f$.
2. Define the breeding function of two $N$-variable boolean functions, $f$ and $g$, as $breed(f,g)$.
3. Generate a set of $T$ random $N$-variable boolean functions, $P_i$ where $i = 1,..,T$.
4. Iterate:
   (a) Generate a set of $\frac{T(T-1)}{2}$ boolean functions, $C_i = breed(P_j, P_k)$, where $j = 1,..,T$, $k = 1,..,T$, $j \neq k$.
   (b) $C'_i = Hillclimb(C_i)$ where $i = 1,..,\frac{T(T-1)}{2}$.
   (c) Form sorted, combined set $S = P \cup C'$, where $S = \{S_1,..,S_{T+\frac{T(T-1)}{2}}\}$ and $fit(S_l) \geq fit(S_{l+1})$, $l = 1,2,..,T + \frac{T(T-1)}{2} - 1$.
   (d) Keep best $T$ functions, by replacing new $P_i = S_i$ where $i = 1,..,T$.
   (e) Reset: optional.
   (f) Iterate until specified stopping criteria is met.
5. Repeat 3. and 4. as required, for a desired number of initial pools.

**Experimental Rationale**

The first use of both techniques together was trialled in [65]. By applying hill climbing to the resultant boolean functions generated by their Genetic Algorithm, the authors were able to make further improvements to the property measures which the Genetic Algorithm was able to achieve individually. Hill climbing was applied subsequent to obtaining the Genetic Algorithm results as the nature of this process is to locally maximize the target property value. Executing this combination of techniques in reverse order would have intuitively meant that no further improvement to a local maximum could be achieved.

The focus of experiments using combined Genetic Algorithm/Hill Climbing techniques involved first applying a Genetic Algorithm to generate a pool of boolean functions whose target cryptographic property measure achievable by the Genetic Algorithm was optimal. Then, each of the boolean functions in the
final pool output by the Genetic Algorithm was used as input to the Hill Climbing Method which endeavoured to locally maximize the desired cryptographic property.

**Previously Reported Results**

An important summary of the experiments which have been conducted using a combined Genetic Algorithm and Hill Climbing Method is now outlined.

An investigation into the validity of combining these two heuristics was first conducted in [65]. Experiments focussed on the best achievable nonlinearity of 8 to 16-variable unbalanced boolean functions generated by combined Genetic Algorithm and hill climbing for samples of 1000 and 10000 functions. These values were contrasted with the same parameters for random generation, independent Hill Climbing Method and independent Genetic Algorithm. For all \( N \) in this range the combined Genetic Algorithm and Hill Climbing Method outperformed each of the other methods in achieving the best nonlinearity.

Experimental trials in [67] concentrated on optimizing the nonlinearity of balanced boolean functions of 8 to 12-variables. A combined Genetic Algorithm and Hill Climbing Method with and without resetting was investigated and best nonlinearities were reported for a poolsize of around 30. These were also directly compared with an independent Genetic Algorithm with and without resetting. Although a combined Genetic Algorithm and Hill Climbing Method consistently yielded higher nonlinearity values than any of the independent Genetic Algorithm tests, the use of resetting in the combined method offered no improvement at all in the nonlinearity values.

**Method Applicability**

Strong evidence that a combined Genetic Algorithm and Hill Climbing Method is superior to either heuristic applied independently has been presented in both [65] and [67]. A pertinent tradeoff to this nonlinearity gain is a performance decrease in terms of computational time.

In Chapter 5 we discuss our application of combining the two heuristics on random generations of \( N \times M \) regular s-boxes with \( N > M \) to improve the nonlinearity property. In addition, the effectiveness and efficiency of the two heuristic techniques applied individually and in combination, are discussed and compared.
3.2 Summary

This chapter has discussed the design rationale and algorithmic process of existing heuristic techniques which will be applied in Chapters 5 and 6 of this thesis to the improvement of substitution boxes in terms of their cryptographic properties. Specifically, the details of two of the earlier existing heuristic methods, Genetic Algorithm and Hill Climbing Method, as well as a method combining the two, have been provided.

The next chapter, Chapter 4, presents three new heuristic methods which have been developed for this thesis. In that chapter, the details and results of the experiments conducted to assess and evaluate the effectiveness of these three new heuristic methods are provided.
Chapter 4

The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

This chapter is devoted to describing the new heuristic techniques or methods which have been developed and applied in this thesis for boolean function property optimization. This includes a description of their algorithms and the types of parameters which are required to be used in their processes. The first method proposes a novel technique to optimize the nonlinearity and autocorrelation measures of balanced even-dimensional boolean functions. The second method is designed to generate highly nonlinear resilient boolean functions by careful manipulation of concatenation layers. The third new method described in this chapter involves a bit changing process dictated by the relationship between specific function transforms and measures in order to generate balanced boolean functions satisfying some non-zero degree of propagation criteria, whilst exhibiting high nonlinearity. Thus, in this chapter, we propose and describe three new heuristic methods, each of which plays a specific role in improving the security properties of cryptographic components of cipher systems.

In this chapter we also discuss in detail the application of each new heuristic method to boolean function property optimization. The aim of applying heuristic
techniques to boolean functions is twofold. Firstly, these techniques have proven (for example in [65], [78], [17]) to be successful in improving the cryptographic properties of boolean functions which, in turn, enhances the security of cipher systems which incorporate such boolean functions. Secondly, heuristic techniques provide an approach to analyzing the boolean function space in terms of factors which include:

- achievable properties and the extent to which combinations of properties may exist together;
- lower and upper bounds on cryptographic property values;
- enumeration of categorized functions;
- lower and upper bounds on enumeration of categorized functions.

The main focus of this chapter is to detail the degree of effectiveness of the new heuristic methods developed for this thesis which have been applied to the optimization of boolean function properties. In particular, we discuss the experimental rationale and justification for each of three new heuristic methods developed for this thesis, as well as explain the appropriate usage of each method. In this chapter we also report on the experimental results achieved by our methods. A number of the best results for each is provided and a discussion on the significance of these results given. This chapter also includes some of the best functions we were able to generate together with the parameters used to achieve these results.

This chapter is appropriately divided into sections according to the main cryptographic property or properties targeted for optimization. The first section focusses on the improvement of the nonlinearity property. The section begins with a discussion on related work in this area performed by other researchers. We then introduce our first new heuristic method which was developed for this thesis to primarily target this property, including a description and the algorithm for the method. The results of the application of this new method to boolean functions are also presented. Further, in this section, a comparison is provided between the results of this new heuristic method and other methods which focus on this property. The second and third sections of this chapter adopt the same content structure as in the first section, with the two further new heuristic methods introduced focussing on resilient boolean functions and boolean functions satisfying some non-zero degree of propagation criteria respectively.
4.1 Highly Nonlinear Cryptographically Strong Boolean Functions

High nonlinearity is one of the most essential properties required, not only for the strength of boolean functions and s-boxes but, also for the security of cipher systems incorporating these components. Examples of cryptanalytic attacks which exploit moderate nonlinearity values are linear cryptanalysis [57] and best affine approximation [26]. In addition to high nonlinearity, balance has long been proven to be an essential property for boolean functions to possess in order to resist a constant function approximation.

Another highly desirable boolean function property is low autocorrelation. This property is commonly measured by the maximum absolute value in the autocorrelation vector (absolute indicator) of a boolean function (see Corollary 2.3) and in [105] is described as being one of the two indicators used in assessing a function’s Global Avalanche Characteristics (GAC). The other GAC measure is called the sum-of-square indicator and, as the name suggests, is computed from the sum of the squares of the values in a function’s autocorrelation vector (see Corollary 2.4). This is also required to be low in order to be a strengthening feature of a boolean function. These avalanche properties are complementary to the nonlinearity of a boolean function as has been discussed in Section 2.1.3 of this thesis. The importance of achieving such properties is perhaps best noted by their ability to resist one of the most powerful cryptanalytic attacks, differential cryptanalysis [4]. The reader is referred to Chapter 2 of this thesis for more information about the abovementioned boolean function properties. Before discussing in detail a new heuristic method which achieves these boolean function properties, we present a brief summary of some other research work (largely algebraic constructions) which target high nonlinearity.

4.1.1 Related Work by Other Researchers

There exists other research work that makes a related contribution to the area of obtaining highly nonlinear boolean functions. This work encompasses research that has been conducted in the construction of such functions largely through algebraic means. A brief discussion of some selected examples of constructions is now outlined.

In [27], Dobbertin provides a proposition for the construction of highly non-
linear balanced boolean functions from normal bent functions. The construction is based on the idea that turning a bent function into a balanced function will ensure high nonlinearity by minimizing the maximum absolute Walsh Hadamard transform value, $WHT_{\text{max}}$ (as defined in Definition 2.17). The substance of the proposition is the modification of the initial segment of a $2N$-variable normal bent function, which by definition is constant, by replacing it with a balanced function whose maximum absolute Walsh Hadamard transform value is small. The effect of constructing a $2N$-variable balanced function, $f$, from a $2N$-variable normal bent function, $g$, and appropriate $N$-variable balanced function, $h$, is expected to be that $WHT_{\text{max}}(f) = 2^N + WHT_{\text{max}}(h)$.

Construction methods for highly nonlinear $N$-variable balanced boolean functions, for even $N \geq 4$, were presented in [92]. Where $N$ is either of the form $N = 4j$ or $N = 4j + 2$, $j \leq 1$, a construction method involving the concatenation of all linear functions (except the first) in $V_2^{2j}$ and $V_2^{2j+1}$ respectively was described. In each case, $l_0(x)$ was replaced with a balanced boolean function. It was then concatenated with the remaining linear functions. For $N = 4j$, $l_0(x)$ in $V_2^{2j}$ was substituted for a balanced $2j$-variable function which is highly nonlinear. For $N = 4j + 2$, $l_0(x)$ was substituted for the function representing the concatenation of the linear functions $l_2(x),..,l_{2j+1-1}(x)$ in $V_2^{2j+1}$. A similar process of concatenating linear functions when $N$ is either of the form $N = 2^i, i \geq 2$, or $N = 2^u(2^v+1)$ $u \geq 1, v \geq 1$, was also outlined in [92] where $l_0(x)$ in each case was obtained by concatenating linear subfunctions to form a bent function but with the leading $k$ bits of the subfunctions replaced by a balanced string, where $k = 4$ in the former case and $k = 2^{2v+1}$ in the latter case. A lower bound on the nonlinearity of balanced boolean functions constructed as such was provided in [92].

In [52] a construction method (Construction 0) for balanced $N$-variable boolean functions, $N > 3$ and odd, was proposed. Theorem 3.1 of [52] proved the low maximum absolute autocorrelation value (absolute indicator) and high nonlinearity provided by this construction. The component functions of this construction were bent. The construction involved the concatenation of two bent functions, $b_1(x)$ and $b_2(x)$, in $V_2^{N-1}$, each produced by the xor sum of bent subfunctions of a specified form. If the hamming weights of $b_1(x)$ and $b_2(x)$ were such that the resulting odd $N$-variable boolean function, $f(x) = b_1(x) \parallel b_2(x)$, was not balanced then $f(x) = b_1(x) \parallel \overline{b_2(x)}$ was used to achieve the balance property.

Other more distantly related construction methods exist, some examples of
4.1. Highly Nonlinear Cryptographically Strong Boolean Functions

which are [103], [45].

4.1.2 Method 1

We now present the first of three methods which we developed to generate boolean functions with specific cryptographic properties. This particular method is used to generate boolean functions which exhibit high nonlinearity, balance and low autocorrelation. Although the target property is nonlinearity, little effort is required to achieve balance and low autocorrelation. The method is hereinafter referred to as “Method 1”.

The basic principle of Method 1 is to concentrate the heuristic search process to regions of the \( N \)-dimensional boolean function space (\( N \) even) that are expected to exhibit high nonlinearity. By definition, these regions comprise functions that are at greater distances from the set of affine boolean functions. The significance of concentrating the search becomes self-evident as we consider larger and larger dimensions of \( N \) input variables.

As with all heuristic methods, particularly for the purposes of efficiency, the first important step is to always begin with a good starting function. This should be one which has reasonably good fitness in the hope that subsequent steps of the method will “discover” better functions with an improved fitness measure. In the case of Method 1, the targeted fitness measure is nonlinearity. Many heuristic techniques simply initiate the process by choosing a starting function at random, or by iteration to a randomly chosen function with a nonlinearity measure above a reasonable value, specified by the code. The approach which we have taken is to indeed begin with a good starting function, although rather than hope to get, or iterate to, a reasonably good starting function, we randomly choose a starting function known to have the maximum nonlinearity value. In the case of an even dimensional input space, this function will be a bent or perfect nonlinear function (see Section 2.1.4). Thus, in Method 1, for even \( N \), we select our starting function randomly from a constructed subset of \( N \)-variable bent functions. In addition to achieving maximal nonlinearity, bent functions have the property of zero autocorrelation. This makes them an obvious choice of starting function when trying to achieve high nonlinearity and low autocorrelation. Note, however, that no bent functions are balanced. As Method 1 makes small incremental changes to the starting bent function, the resultant boolean function from Method 1 possesses a similar structure and like characteristics to the chosen starting bent
function.

There has been a significant amount of research performed on the topic of bent functions, in particular on methods of constructing bent functions. Perhaps one of the earliest and best known constructions of bent functions is referred to as the Maiorana-McFarland generalization of bent functions, based on the independent research of both Maiorana (unpublished, see [25]) and McFarland [59]. This class of \( N \)-variable bent functions is constructed from \( \frac{N}{2} \)-variable boolean functions. An alternative method of construction was discovered by Dillon in 1975 [25] which produced a different class of bent functions. It is worth noting that this earlier work did not use the term “bent” function. It was not until Rothaus in 1976 [82] that classes of functions exhibiting unique characteristics were collectively identified using the generic term, “bent” functions. There have been many new construction methods obtained by modifications to the above constructions. These have led to a small number of new classes of bent functions being discovered, for example, [8] and [104].

In Method 1 we use a Maiorana-McFarland-like construction of bent functions throughout the computation, however, any construction of \( N \)-variable bent functions (\( N \) even) may be used. Since the starting function/s for Method 1, \( N \) even, are bent, they have a hamming weight of \( 2^{N-1} \pm 2^{\frac{N}{2}-1} \). As balance is an important property for cryptographic applications, we seek to achieve balance while retaining high nonlinearity and low autocorrelation. In Method 1, this is done by progressively setting or clearing \( k \)-bits at a time, and retaining resultant functions which remain above a certain nonlinearity threshold. Note that it is possible to obtain \( 2^N \) boolean functions of hamming distance 1 from any \( N \)-variable boolean function, and recall that a one bit change in the truth table of a boolean function results in a change to the Walsh Hadamard transform of the function \( \in \{+2,-2\} \) (see Lemma 2.7). This in turn produces the effect of a small increase or decrease to the nonlinearity of the function. The key factor of Method 1 is the generation of boolean functions with a specified hamming distance from the original starting function and subsequently from retained functions. By doing so, we are maintaining much of the distance needed to provide us with boolean functions with high nonlinearity and low autocorrelation. The degree of bit changing, which depends on the imposed hamming distance/s, allows a branching effect to take place.

Let \( L \in \{0,...,\frac{2^N-1}{k}\} \) denote the level of changes in hamming distance (changing by \( k \) bits) from the original starting function. Thus, for the starting function, \( L \)
4.1. Highly Nonlinear Cryptographically Strong Boolean Functions

Maxchanges is defined to be the maximum number of distinct \( k \)-bit changes to be considered at each level \( L \). Note that our experiments using Method 1 were performed by choosing \( k = 2 \). Maxchanges is fixed for each run of the program and its most optimal value may be determined by experimentation. However, typically maxchanges took a random value for each program run, but often chosen to be a random value approximately between \( 2^{N-1} \) and \( 2^{N+1} \). \( NL_{\text{min}} \) is defined as the minimum nonlinearity value which is acceptable for the resulting boolean function. A sample size may be specified for the code, samplesize. The algorithm for Method 1 is set out in Algorithm 4.1.

The key differences between Method 1 and the Hill Climbing Method (Algorithm 3.1) which utilizes two bit changes in balanced functions are as follows:

- The initial functions used in Method 1 are bent functions as compared to random balanced functions to start the hill climbing process.
- Method 1 commences with the generation of a set of sample bent functions from which the initial function is randomly selected, while the Hill Climbing Method generates one random balanced function at a time for processing.
- Method 1 seeks to achieve balance, whilst the Hill Climbing Method simply maintains balance.
- Each level in Method 1 represents a two bit cumulative hamming weight difference from the starting bent function. The Hill Climbing Method only has one “level” represented by a two bit change from the original function, but always with balanced weight.
- The number of levels in order to reach balance in Method 1 is fixed at \( \frac{2^N-1}{k} \). For the Hill Climbing Method, the starting function is always balanced and a repeated move to only a single level still maintains this balance.
- Whilst trying to achieve balance, the goal of Method 1 is to ensure that nonlinearity doesn’t fall below a specified threshold value. In contrast, the goal of the Hill Climbing Method is to maximize (increase) nonlinearity as much as is possible, while maintaining the balance property.
- Method 1 permits backtracking to previous levels at all times if a particular set of accepted changes leads to functions below the threshold, while the Hill Climbing Method does not support any backtracking once a modified function has been accepted. The Hill Climbing Method only enables backtracking to the previous function if the proposed bit complementation is not accepted.
- The backtracking to previous levels inherent in Method 1 permits many
Algorithm 4.1: Method 1

1. Let $\Lambda$ be a constructed subset of $N$-variable bent functions.
2. Select a random function $r(x), x = (0,\ldots,2^N - 1)$, where $r(x) \in \Lambda$.
3. Define $T0$ as the set of positions $i$, where $r(i) = 0$, $T1$ as the set of positions $j$, where $r(j) = 1$, and $hw(r)$ as the hamming weight of $r(x)$.
4. Call the REPLACE function with $L = 0$ and let $f(x) = r(x)$.
5. If $\text{samplesize}$ not reached, return to Step 2.
6. If $\text{samplesize}$ reached, exit program.

REPLACE function (function $f(x)$, level $L$):

1. Check $hw(f)$:
   
   (a) If $hw(f) < 2^{N-1}$, select $a_1 \in T0$, $a_2 \in T0$, where $a_1 \neq a_2$.
   
   (b) If $hw(f) > 2^{N-1}$, select $a_1 \in T1$, $a_2 \in T1$, where $a_1 \neq a_2$.
   
   (c) If $hw(f) = 2^{N-1}$ (this only occurs if $hw(r) \pm 2L = 2^{N-1}$):
      
      i. If selection criteria met, output $f(x)$, $NL(f)$, $AC_{\text{max}}$ and store function. Go to Step 6 of REPLACE function.
      
      ii. If selection criteria not met, go to Step 6 of REPLACE function.

2. Derive candidate function $g(x)$ from $f(x)$, with $f(a_1) = f(a_1) \oplus 1$; $f(a_2) = f(a_2) \oplus 1$.

3. Check $NL(g)$.

4. If $NL(g) \geq NL_{\text{min}}$, call the REPLACE function recursively with $L = L + 1$ and $f(x) = g(x)$.

5. Discard $g(x)$.

6. (a) If $\text{maxchanges}$ has not been reached, return to Step 1 in REPLACE function. $L$ does not change.

   (b) If $\text{maxchanges}$ has been reached, and $\text{samplesize}$ has not been reached, return to Step 1 in REPLACE function with $L = L - 1$, else exit.
good functions to be generated for little incremental effort, while the Hill Climbing Method is significantly less efficient as generating more than one function requires proportionally additional effort.

- Due to the starting functions for Method 1 being maximal for nonlinearity, and the small fixed movements away in hamming distance, Method 1 is more likely to achieve higher nonlinearities than the Hill Climbing Method.
- Method 1 is restricted to the even dimensional boolean function space, whereas the Hill Climbing Method can be used in even or odd dimensional spaces.

Method 1 is a useful and efficient method for generating highly nonlinear balanced boolean functions which exhibit low autocorrelation. Method 1 represents a new heuristic technique based on the simple idea that an incremental move in the direction of balance from a bent function, whilst incorporating careful selection of bit changes in accordance with nonlinearity observations, will maintain an acceptable distance from the closest affine function.

Independently from the work reported in this section, in 1989 [62] a strategy for finding nearly perfect nonlinear balanced boolean functions was suggested where an arbitrary set of $2^{N-1}$ bits of a perfect nonlinear function may be complemented in order to make the function balanced. The strategy subsequently involved the idea of generating functions with a minimum nonlinearity but noted that random or exhaustive search was unlikely to be computationally feasible in achieving the desired functions. However, no method, experiments nor results from this suggested strategy were reported.

The application of Method 1 to $N$-variable bent boolean functions ($N$ even) in order to generate balanced $N$-variable boolean functions with high nonlinearity and low autocorrelation is now described below. We also display the results of experiments conducted for a range of input sizes and discuss the effectiveness of this approach.

**Experimental Rationale**

The algorithm for Method 1, Algorithm 4.1, is described above. The starting functions for the Method 1 algorithm are bent and thus the method operates solely in the even-dimensional boolean function domain. The method was developed to require bent functions as input due to two key properties which they characteristically possess - all bent functions exhibit maximal nonlinearity and zero autocorrelation. The properties which Method 1 was designed to achieve
are balance, high nonlinearity and low autocorrelation. Due to the utilization of bent functions to start the computational process, the method is limited to the even-dimensional search space and also to these specific properties as other cryptographic properties such as correlation immunity are unsuitable for optimization by this algorithm. In the case of correlation immunity, its characteristics directly conflict with those of bent functions in terms of Walsh Hadamard transform distribution and therefore cannot successfully be targeted by this method.

The focus of experiments with Method 1 have been to efficiently discover, through directed search, the best achievable even $N$-variable boolean functions which are both balanced and highly nonlinear. Although the method provides no mechanism for directly minimizing the maximum absolute autocorrelation value of the resulting boolean functions, these values have been consistently observed to be low. Low autocorrelation is a consequence of initiating the computation with bent functions which have zero autocorrelation and the expectation that small movements away from a bent function will result in small movements away from zero autocorrelation. Thus, low autocorrelation is achieved by Method 1 “for free”. Although a similar argument holds for nonlinearity, the Method 1 algorithm has incorporated acceptance/rejection criteria for nonlinearity which enhances the achievement of high nonlinearity goals.

**Experimental Results**

We now present a collection of results from experiments performed using Method 1. Particular testing using this method included:

(i) balanced boolean function generation recording highest nonlinearity achievable for each even $N \in \{6,8,10,12,14\}$;

(ii) balanced boolean function generation recording optimal combinations of highest nonlinearity and lowest sum-of-square indicator for each even $N \in \{6,8,10,12,14\}$;

(iii) balanced boolean function generation recording optimal combinations of highest nonlinearity, algebraic degree, maximum absolute autocorrelation value and sum-of-square indicator for each even $N \in \{6,8,10,12,14\}$.

Table 4.1 below shows the highest nonlinearity values which have been achieved for balanced $N$-variable boolean functions $N \in \{6,8,10,12,14\}$ for Method 1. The table also shows, for comparison purposes, the highest nonlinearity values that were achieved by a number of other heuristic methods which are examples of the previously best known heuristic results for these properties at the time the work
reported in this section was performed. The results of these methods are directly compared in the table with Dobbertin’s *conjectured* values [27] for maximum non-linearity achievable by balanced even-dimensional boolean functions. Note that a dash in the table indicates that no result was reported.

<table>
<thead>
<tr>
<th>N</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL - Method 1</td>
<td>26</td>
<td>116</td>
<td>488</td>
<td>2002</td>
<td>8104</td>
</tr>
<tr>
<td>NL - [65], [67]</td>
<td>26</td>
<td>116</td>
<td>484</td>
<td>1980</td>
<td>8036</td>
</tr>
<tr>
<td>NL - [16]</td>
<td>26</td>
<td>116</td>
<td>484</td>
<td>1990</td>
<td>-</td>
</tr>
<tr>
<td>NL - [17]</td>
<td>26</td>
<td>116</td>
<td>486</td>
<td>1992</td>
<td>-</td>
</tr>
<tr>
<td>NL conjectured in [27]</td>
<td>26</td>
<td>116</td>
<td>492</td>
<td>2010</td>
<td>8120</td>
</tr>
</tbody>
</table>

Table 4.1: Highest nonlinearity found by various heuristic methods for balanced $N$-variable boolean functions, $6 \leq N \leq 14$, $N$ even, compared with Dobbertin’s *conjectured* maximal nonlinearity values

The results of [65] and [67] were achieved by combinations of Genetic Algorithms and hill climbing techniques in 1997 and 1998 respectively. At that time these were the best known results by any heuristic method and compared favourably with benchmark values obtained by random search as well as individual Genetic Algorithm results. In 2000, results obtained using simulated annealing followed by hill climbing in [16] produced equally effective results with a slightly better result for 12-variable functions. In 2002, the results were again improved in [17] by a Nonlinearity Targeted Approach (NLT) based on simulated annealing. Our subsequent 2002 results for Method 1 have further improved on all these methods with respect to $N \in \{10, 12, 14\}$. The Method 1 results shown in Table 4.1 represent the closest discovered values using heuristic methods to Dobbertin’s conjectured maximal nonlinearity values [27] for even dimensional balanced boolean functions of these inputs. The maximal nonlinearity for $N$-variable balanced boolean functions is not definitely known for $N \geq 8$. In particular, an interesting open question for some time has been whether 8-variable balanced functions with nonlinearity 118 exist or not. The reader should note that, to date, no counter-examples disproving Dobbertin’s 1994 conjecture have been found.

Table 4.2 provides the highest nonlinearity and lowest sum-of-square indicator combination of properties achieved by Method 1 for even $N$, $6 \leq N \leq 14$.

Optimum combinations of nonlinearity and sum-of-square indicator exhibited by boolean functions contribute to their resistance to linear cryptanalysis whilst,
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

Table 4.2: Best values for combined nonlinearity ($NL$) and sum-of-square indicator ($\sigma$) measures exhibited by even-dimensional balanced boolean functions generated by Method 1

<table>
<thead>
<tr>
<th>$N$</th>
<th>$NL$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
<td>6,784</td>
</tr>
<tr>
<td>8</td>
<td>116</td>
<td>86,656</td>
</tr>
<tr>
<td>10</td>
<td>488</td>
<td>1,262,464</td>
</tr>
<tr>
<td>12</td>
<td>2000</td>
<td>18,743,680</td>
</tr>
<tr>
<td>12</td>
<td>2002</td>
<td>18,754,048</td>
</tr>
<tr>
<td>14</td>
<td>8100</td>
<td>284,866,432</td>
</tr>
<tr>
<td>14</td>
<td>8102</td>
<td>284,891,392</td>
</tr>
<tr>
<td>14</td>
<td>8104</td>
<td>284,919,808</td>
</tr>
</tbody>
</table>

at the same time, providing good diffusion. At the time this work was performed, there had been no prior published values that directly associated high nonlinearity and minimal sum-of-square indicator. Table 4.2 above provides this association for $N \in \{6,8,10,12,14\}$. Prior to this work, in 2002, [17] reported on the sum-of-square values achieved by their simulated annealing technique for $5 \leq N \leq 10$, but did not consider associated nonlinearity values in their findings. Regardless, even with the additional constraints placed on the Method 1 results of Table 4.2, the sum-of-square indicator values achieved by Method 1 equalled the best reported for $N \in \{6,8\}$ and surpassed the reported $N = 10$ value.

Trials of Method 1 successfully generate many examples of balanced $N$-variable highly nonlinear functions with low autocorrelation and maximal algebraic degree $N \in \{6,8,10,12,14\}$. Some of the best examples of combinations of property measures exhibited by boolean functions which we have generated using Method 1 are shown in Examples 4.2 to 4.5.

Example 4.2: 6-variable boolean function (in hex notation) with $NL = 26$, $deg = 5$, $AC_{max} = 16$ and $\sigma = 6,784$

a4ebd7d8b0b6808d

Table 4.3 contains a list of some of the best combined property measures of nonlinearity, algebraic degree, absolute indicator and sum-of-square indicator values for functions produced by Method 1. Good values for this combination of properties is highly sought for resistance against both differential and linear
Example 4.3: 8-variable boolean function (in hex notation) with $NL = 116$, $deg = 7$, $AC_{\text{max}} = 16$ and $\sigma = 89,728$

7eb4719b4da742a8bbe124ce18fa17fd7e6b716c4d58c2572b3e3431180d1702

Example 4.4: 10-variable boolean function (in hex notation) with $NL = 488$, $deg = 9$, $AC_{\text{max}} = 40$ and $\sigma = 1,272,448$

b024e36e57b6a3bf2bece9ea7eb9bc8317d0d5d64285808c18cfd8d94d8a8fb0
dbe311e58eb64c6fd4ec06ea81b94383e8d02ad6bd857f0ce75f259d928a70b024
1c19e571494cbf2b1306ea7e6438317272ad6427a7d8c1820259d47570b0db1c
e6e50e49b3bfd413e9ea8146bc82e82fd556bd7a808ce720dad992758f

Example 4.5: 12-variable boolean function (in hex notation) with $NL = 2000$, $deg = 11$, $AC_{\text{max}} = 64$ and $\sigma = 18,757,120$

2c63a66c3ce83d0d7936f33969bde8582c63a66cc317c2f2793ef33964297a723
6ca96333e732027639fc3666b26757236ca963cc18cfd7639fc36994d9ca82c9c
a6933c1f3df279c9f3c6694268a72c9ca693c3e8c20d79c9f3c6696bd97582393a9
9c331832fd76c6fcc9664d67a82393a99ccce7cd0276c6fcc999b29cd72c635993
3ce8e2f279360cc669bd97a72c635993c3173d0d79360cc696426858236c569c33
e7c2fd763903c9666b298a8236c569cc183202763903c9994d675fc9clbc3c17
c20d79c90c3969429f582c9c596cc3e83df279c90e3996bd68a7239356633318cd
0277d6033666d9857239b5663ce732fd77c6033669b267a81f50955f0f0db8e3e
4a17cc0a5a8eb6b1f50955f0f024f1c14a15c00aa57ia494059f9a51204013145
0acf055581546c105f9a70f2bfce450acfc25aa7eab9b1f2f59a0f0f490ec14a1fa
c0f55a715b941fa95a0f0dfb1f4e4a6c0f5a58ea46b10a9aaf0b01ce45f5c8c
a575e5498f00a9aefdf4c3f17f5ccfbaaa81ab641f506a00f00f1c14b053ff5
5a8ea4941f506aa0f2240e3ae4a053ff5a5715b6b105f65af00d4f2ece450b30fa55
81ab9b105f5afff2b013145a03b4faa7ea5461fa6a5f0f24f1be4e3f0a5a71
a46b1fa6a5ff0f0db0ec14afa3f0a58eb59b410b0655002bfe3145f5b005557eab6
410a06550f0d401ce45f53005aa81549b

cryptanalysis whilst maintaining high algebraic complexity. We now compare the
results in Table 4.3 with previous work in this area.

In 2000, the authors of [16] used the heuristic techniques of simulated annealing
and combined simulated annealing and hill climbing to produce the then best known
results of balanced boolean functions with high nonlinearity and low auto-
correlation. The property values reported in [16] can be directly compared to the
nonlinearity and absolute indicator results in Table 4.3 produced by Method 1. For $8 \leq N \leq 12$, $N$ even, we have found a great number of $N$-variable balanced boolean functions with consistently lower $AC_{max}$ values for identical values of nonlinearity. In addition, Method 1 is capable of finding functions which achieve lower $AC_{max}$ values with higher nonlinearity than in [16]. A further comparison can be made between the maximum nonlinearity they were able to achieve for balanced functions for $6 \leq N \leq 12$, $N$ even, and the highest nonlinearity of the balanced boolean functions Method 1 was able to generate. For each number of even inputs $N$ in this range, Method 1 either achieved an equal nonlinearity value or a higher nonlinearity value.

Explicit results for 6 and 15-variable balanced boolean functions were reported in [53] in 2002 with respect to nonlinearity, $AC_{max}$, sum-of-square indicator and algebraic degree. This was achieved by the use of previously published initial functions with good characteristics which formed the basis for computer searches, together with the application of a linear transformation at the conclusion of the search. Their best example functions also satisfied SAC (see Definition 2.21). The 6-variable balanced boolean functions that were generated by Method 1 exhibited property measures that were equal to [53] with respect to nonlinearity and algebraic degree and represented an improvement on the properties of absolute indicator ($AC_{max}$) and sum-of-square indicator. In addition, Method 1 was able to generate a large number of these functions satisfying SAC without the need to invoke a linear transformation to achieve this, as was required in [53]. We provide below in Example 4.6 an example truth table (in hex notation) of a 6-variable

<table>
<thead>
<tr>
<th>$N$</th>
<th>$NL$</th>
<th>$deg$</th>
<th>$AC_{max}$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
<td>5</td>
<td>16</td>
<td>6,784</td>
</tr>
<tr>
<td>8</td>
<td>116</td>
<td>7</td>
<td>16</td>
<td>89,728</td>
</tr>
<tr>
<td>10</td>
<td>488</td>
<td>8</td>
<td>40</td>
<td>1,268,608</td>
</tr>
<tr>
<td>10</td>
<td>488</td>
<td>9</td>
<td>40</td>
<td>1,272,448</td>
</tr>
<tr>
<td>12</td>
<td>2000</td>
<td>11</td>
<td>64</td>
<td>18,757,120</td>
</tr>
<tr>
<td>12</td>
<td>2002</td>
<td>11</td>
<td>64</td>
<td>18,776,704</td>
</tr>
<tr>
<td>14</td>
<td>8100</td>
<td>13</td>
<td>104</td>
<td>284,931,328</td>
</tr>
<tr>
<td>14</td>
<td>8102</td>
<td>13</td>
<td>104</td>
<td>284,891,392</td>
</tr>
<tr>
<td>14</td>
<td>8104</td>
<td>13</td>
<td>112</td>
<td>284,919,808</td>
</tr>
</tbody>
</table>

Table 4.3: Best measures of property value combinations exhibited by even-dimensional balanced boolean functions generated by Method 1
4.1. Highly Nonlinear Cryptographically Strong Boolean Functions

boolean function generated by Method 1 with the same highest achievable nonlinearity and optimal degree but with better absolute indicator and sum-of-square indicator values than were reported in [53].

Example 4.6: 6-variable balanced boolean function with $deg = 5$, $NL = 26$, $AC_{max} = 16$, $\sigma = 6784$ and satisfying $SAC$

04511b5e37e23e6d

As Method 1 was only applicable for even-dimensional functions, no comparison with the reported 15-variable function in [53] was possible.

Further high quality property results were obtained in 2002 [17] by combining simulated annealing, hill climbing and linear transformation. From Table 3 in Section 3 of their paper we are able to compare their combined balance, nonlinearity, degree, and $AC_{max}$ properties for $6 \leq N \leq 12$ ($N$ even) with the results obtained from our Method 1 in Table 4.3. For all even $N$ in this range, Method 1 was able to at least equal their results for these properties, and for most values of $N \geq 8$ could improve on both the nonlinearity and $AC_{max}$ values.

In 2003, results of simulated annealing searches arising from the improvement of the cost function utilized in [16], [17], were reported in [44]. The cryptographic properties targeted by the simulated annealing algorithm were nonlinearity, algebraic degree and maximum absolute autocorrelation value (which formed part of their “profile”), and where possible, balance. The number of input variables in which experiments were conducted were for $N \in \{8, 9, 10, 11\}$. As the Method 1 trials were conducted for $N$ even, in particular, $N \in \{6, 8, 10, 12, 14\}$, direct comparisons may only be made for 8 and 10-variable boolean functions. Both Method 1 and the simulated annealing method of [44] were able to generate 8-variable 7th degree balanced boolean functions with nonlinearity of 116, which is the highest known. However, a lower $AC_{max}$ value was easily achievable by a large number of $N = 8$ functions produced by Method 1, with this same high nonlinearity value and algebraic degree. A comparison between the 10-variable functions generated by each method shows that Method 1 was able to obtain multiple balanced boolean functions with both higher nonlinearity values combined with lower $AC_{max}$ values than achievable in [44], for the same optimal algebraic degree.

The computational effort required by Method 1 is favourable when compared
to other heuristic methods. This is largely due to the small number of bit changes
\((\pm 2^{\frac{N}{2}} - 1)\) required to transform a bent function into a balanced function. In general, Genetic Algorithms are computationally intensive due to the sorting component necessary in the process. Algorithms based on hill climbing are quicker than Genetic Algorithms partly as no sorting is involved. Method 1 uses a recursive branching structure with a mechanism for backtracking to functions which have been partially investigated and may still prove to be potentially useful for achieving the desired nonlinearity. These features of the method help to minimize the computational effort involved in generating multiple boolean functions with the required properties. Thus, in general, Method 1 represents a speedier alternative to hill climbing as hill climbing allows backtracking to the previous function only and its process may only achieve one resulting boolean function for each single starting function. In addition to generally being more efficient than the Hill Climbing Method, Method 1 has shown its ability to achieve better property values, particularly for larger \(N\). Method 1 also has the advantage of providing the extra property of low autocorrelation with no additional effort. The table below contains average run times for the execution of the Method 1 program for each \(N \in \{6, 8, 10, 12, 14\}\). These run times reflect program execution until the desired number of sample good functions have been obtained. Typically, this has been in the order of tens of good functions for the times stated for larger \(N\) \((N \geq 12)\) and hundreds of good functions for the times stated for smaller \(N\) \((N \leq 10)\).

\[
\begin{array}{|c|c|c|c|c|}
\hline
N & 6 & 8 & 10 & 12 & 14 \\
\hline
\text{Computational program run times O(seconds)} & 0 & 0 & 10 & 60 & 1800 \\
\hline
\end{array}
\]

Table 4.4: Order of magnitude of program run times for Method 1 to achieve \(N\)-variable boolean functions with \(6 \leq N \leq 14\), \(N\) even.

Table 4.4 shows that the average program run times for Method 1 to achieve balanced boolean functions with high nonlinearity and low autocorrelation ranges from approximately 0 seconds for 6-variable inputs to approximately 30 minutes for 14-variable inputs. The majority of experimental runs for Method 1 were performed on a Pentium III 700 MHz PC (typically for \(N \leq 8\)) and a Pentium III 1000 MHz PC (typically for \(N \geq 8\)). As comparable published experiments, as cited earlier in this section, have not reported on their run times, no comparison is possible.
4.2. Resilient Boolean Functions

Method Applicability

Method 1 successfully generates highly nonlinear, low autocorrelation, even-dimensional balanced boolean functions for $6 \leq N \leq 14$, $N$ even, some of which are the best known for these combination of properties. Computationally, experiments with Method 1 have clearly demonstrated the operational efficiency of generating a large number of good boolean functions with the desired characteristics. Due to this inherent efficiency, the application of Method 1 to larger even-dimensional boolean functions is a feasible future avenue of research.

Method 1 is capable of easily generating boolean functions satisfying $SAC$ without needing to resort to a further process of linear transformation. Further, the ability to achieve functions with good $GAC$ indicators enables strong resistance against differential cryptanalysis. Similarly, boolean functions generated by Method 1 may be combined to form strong $s$-boxes with the ability to resist differential cryptanalysis. The absence of linear structures in functions generated by Method 1 is another desirable consequence of the Method 1 generation process.

Because of the requirement that the starting functions are bent, Method 1 is limited to the even-dimensional input space. For this reason also, Method 1 implicitly targets the search sub-space containing high nonlinearity, low autocorrelation functions. Future research directions involving or extending Method 1 could concentrate on experimentation to focus on achieving additional complementary target properties such as propagation criteria.

4.2 Resilient Boolean Functions

Resilience is an essential cryptographic property for boolean functions which are incorporated into the types of cipher systems whose most significant source of strength relies on little or no correlation between the combined input bits and the output bits of its component functions. The most common utilization of resilient boolean functions has been in stream ciphers, particularly as combining functions for linear feedback shift registers. Cryptanalytic attacks on stream ciphers typically focus on revealing the secret key by retrieving the initial states of the linear feedback shift registers. This task is made easier if a high correlation exists between the input and output bits of the combining boolean function. The types of attacks which exploit this correlation are termed correlation attacks (for example, original [95] and fast [61], [41], [40]). Correlation attacks are discussed
in more detail in Section 2.3.3 of this thesis.

As discussed in Section 2.1.3 of Chapter 2, there exists a tradeoff between the correlation immunity and nonlinearity properties: the higher the order of resilience (or correlation immunity) of a function, the lower the nonlinearity of that function. Before discussing a new heuristic method for generating functions which achieve a good balance between these two conflicting properties, we present a brief summary of other techniques that have been used to produce functions with good correlation immunity properties.

### 4.2.1 Related Work by Other Researchers

A significant amount of research work has been done in the area of correlation immune and resilient boolean functions. In particular, the obtaining of such functions using methods which are not, in all or part, predominantly heuristic, can be considered to be partially related to the work in this thesis on this topic. This section aims to outline just some of the other proposed methods for obtaining such functions.

A method was proposed in [24] for constructing $N$-variable balanced boolean functions to achieve a non-zero order of correlation immunity coupled with high nonlinearity. The construction represented a combination of heuristic techniques and an algebraic construction method. The process involved the use of a combined Genetic Algorithm/Hill Climbing Method to first generate starting balanced subfunctions with high nonlinearity. The construction portion of the process was from [102] encompassing a subfunction transformation using a generating matrix of an $[N,k,d]$ linear code [18], which guaranteed a minimum order of correlation immunity in the constructed function. Results for 10-variable balanced boolean functions were provided in [24] and comprised the obtaining of 10-variable balanced CI(1) boolean functions with nonlinearity 480, and 10-variable balanced CI(2) boolean functions with nonlinearity 464.

A construction method was proposed in [51] to produce unbalanced CI(1) even-dimensional boolean functions. This construction required a lower dimensional (i.e. $N-2$) unbalanced CI(1) starting function with algebraic degree $> 2$. This starting function, $f$, was then concatenated in the form $f \parallel f \parallel f \parallel \bar{f}$, which produced an $N$-variable unbalanced CI(1) function with nonlinearity $2^N + 2NL_f$. A further modification was also proposed, by complementing two bits, $i$ and $2^{N-1-i}$, to produce an unbalanced CI(1) function with algebraic degree $N - 1$. Such
4.2. Resilient Boolean Functions

A modification relies on a palindromic relationship existing between the starting and constructed function and a condition on the degree of the starting function, \( f \).

A specific non-generalized construction method for an 8-variable and a 10-variable 1-resilient boolean function with nonlinearity 116 and 488 respectively was proposed in [55]. This was based on complementing certain specified bits of an \( N \)-variable bent function of a certain form for \( N = 8 \) and 10 respectively. The 8-variable function construction encompassed the complementing of all output bits of the utilized bent function where the weight of the input to this function was \( \in \{0,1,N\} \). Similarly, the single 10-variable function was constructed in this fashion but with a different defined sequence being complemented in the bent function’s output. Thus, it was shown that complementing a specified sequence of bits of a particular bent function resulted in balance being achieved as well as first-order correlation immunity for these two input sizes. No general method for choosing the required sequence of bits was given.

A small extension to [55] was reported in [56] where it was shown that any Maiorana-McFarland bent function that conformed to certain requirements was able to be used as a starting function. These requirements were defined for 8-variable boolean functions, and an application of the construction (now generalized) resulted in balanced, first-order correlation immune functions with a nonlinearity of 116.

In 2002, a review of the then current state of the research in the area of correlation immune boolean functions was compiled in [83]. While no new advances were reported, [83] represents a useful taxonomy of the research work which had been performed in this area up to that time. The reader is referred to [83] for this interesting review which includes a summary of construction methods and bounds on cryptographic properties achievable for correlation immune functions.

Some examples of other research work in constructing \( CI(m) \) and/or \( t \)-resilient boolean functions, though of reduced relatedness to the work in this thesis, can be found in [99], [98], [28], [45] and [33]. We also refer the reader to some of the research work which has been performed on the construction of multiple output functions (s-boxes) which are \( CI(m) \) or \( t \)-resilient. These include [106], [47], [42] and [35].

In the next section we discuss a new method for generating functions which achieve a good balance between these two conflicting properties.
4.2.2 Method 2

In this section, we describe a method which we have designed for generating optimized resilient boolean functions. Optimality is defined as the best known combination of balance, high nonlinearity and order of correlation immunity, together with an algebraic degree which maximizes Siegenthaler’s inequality (see Theorem 2.5). This method is referred to as “Method 2”.

For Method 2, we operate in the Walsh Hadamard transform domain. This enables us to force the generation of functions that satisfy correlation immunity goals. Operating in the Walsh Hadamard transform domain also enables direct limiting of maximum absolute values within the Walsh Hadamard transform vector, which has a direct relation to nonlinearity.

It is well known that the concatenation of two valid Walsh Hadamard transform vectors of dimension $N$ results in a valid Walsh Hadamard transform vector of dimension $(N+1)$ (see Definition 2.14). The building of higher dimension functions using step-by-step concatenation of their Walsh Hadamard transform vectors forms the basis of Method 2. To determine whether or not a newly concatenated boolean function is to be kept or discarded, we apply several selection criteria to it. These selection criteria ensure that the kept boolean functions for each level of $N$ possess the particular minimum property measures selected to ensure that the target $N$ balanced boolean functions are able to be constructed with the desired combinations of property values for nonlinearity and correlation immunity. Selection criteria are pre-determined to enable the best combination of property values to be obtained.

The selection criteria used at each level are as follows:

- **Maximum absolute Walsh Hadamard transform vector value ($WHT_{maxN}$):** used to enforce a minimum nonlinearity at each $N$ level.

- **Minimum absolute non-zero Walsh Hadamard transform vector value ($WHT_{minN}$):** at times, used to restrict the range of possible Walsh Hadamard transform vector values.

- **Minimum order of correlation immunity ($CI_N$):** used to enable easier generation of higher order correlation immune functions for larger $N$. It should be noted that the code for Method 2 automatically incorporates the balance property for any order of correlation immunity $\geq 1$. 
It is a trivial exercise to generate a complete list of all 4-variable boolean functions and their characteristics. We use a selection of 4-variable boolean functions as the starting pool for Method 2. The selection criteria listed above may be defined independently for boolean functions, \( f \) and \( g \), at each dimension \( N \), \( N = 4 \) to \( \text{target} N \). The Method 2 algorithm is described in Algorithm 4.7.

Algorithm 4.7: Method 2

1. Let \( L_4 \) be a set of \( T \) \( N = 4 \) boolean functions that satisfy \( \text{WHT}_{\text{max}} \), \( \text{WHT}_{\text{min}} \), and \( \text{CI} \). Let \( R_4 \) be a set of \( T \) \( N = 4 \) boolean functions that satisfy \( \text{WHT}_{\text{max}} \), \( \text{WHT}_{\text{min}} \) and \( \text{CI} \).

2. For \( N = 5 \) to \( \text{target} N \)
   
   (a) Call the BUILD procedure(\( N, L_N \))
   
   (b) Call the BUILD procedure(\( N, R_N \))

3. Perform an inverse Walsh Hadamard transform on each of the final \( \text{target} N \)-variable functions to determine their truth tables from the concatenated \( \text{WHT} \).

BUILD procedure(\( N, S_N \)):

1. Select \( f(x) \) where \( f(x) \in L_{N-1} \)

2. Select \( g(x) \) where \( g(x) \in R_{N-1} \)

3. Concatenate the \( \text{WHT} \)s of \( f(x) \) and \( g(x) \) to form the \( \text{WHT} \) of an \( N \)-variable boolean function, \( h(x) \)

4. Add \( h(x) \) to the set \( S_N \) iff \( h(x) \) satisfies \( \text{WHT}_{\text{max}} \), \( \text{WHT}_{\text{min}} \) and \( \text{CI} \).

5. Return to Step 1. until the set \( S_N \) is of the desired size.

In addition to the imposed selection criteria, the set sizes at each level \( N \) are specified by the code. Clearly, it is more useful, to decrease the set size as \( N \) increases and tends to \( \text{target} N \), as the number of functions satisfying the selection criteria at each dimension \( N \) will be fewer than the number of functions in the set.

In Steps 1. and 2. of the BUILD procedure in Algorithm 4.7, a selection
process to choose a function from the set occurs. Among the different selection processes which have been trialled for Method 2 are random selection and exhaustive pairing. Each of these processes is described below.

The random selection process allows a boolean function to be chosen randomly from each of the sets $L_{N-1}$ and $R_{N-1}$. These two chosen boolean functions are concatenated to form an $N$-variable boolean function, which is retained if the selection criteria are met. Our experimental trials of Method 2 always employ random selection for at least $N = 5$ and $6$ so that non-deterministic factors influence the computation.

The exhaustive pairing process may be used in the higher levels of Method 2 to ensure that all distinct pairings of boolean functions at $N - 1$ are tested for satisfaction of the selection criteria. This is a practical approach at this stage of Method 2 since the retained sets of boolean functions for higher $N$ become smaller and exhaustive concatenation of pairs is not too computationally intensive. We typically used this process for the $targetN$ and $targetN - 1$ variable levels.

The elegance of this method is the ease by which the normally conflicting properties of nonlinearity and correlation immunity can be considered simultaneously. This is a novel approach that capitalizes on the process of concatenation, often used for algebraic construction, in a heuristic that achieves the best trade-off between two conflicting properties.

We discuss below our application of Method 2 to $N$-variable boolean functions, $N = 4$ to $targetN - 1$, which satisfy our selected criteria, in order to progressively generate $targetN$-variable resilient boolean functions with high nonlinearity. The concatenation aspect of the method enables the construction of higher dimensional Walsh Hadamard transform vectors which obey Parseval’s Equation (see Theorem 2.2) from lower dimensional Walsh Hadamard transform vectors. The concatenation can be applied recursively leading to significantly higher dimensional boolean functions being built upon the foundations of more easily generated lower dimensional functions exhibiting good measures in the desired cryptographic properties. Below we also present the results of the method with respect to optimum combinations of desired properties and give examples of selected functions, as well as specifics of the parameters which generated them. The efficiency and effectiveness of Method 2 at each level, $N$, is now discussed, together with the limitations of the method.
Experimental Rationale

Method 2 was designed for the generation of balanced correlation immune boolean functions with high nonlinearity. The resilience and high nonlinearity properties are achieved simultaneously from the selection criteria which has been built into the algorithm. The Method 2 algorithm, Algorithm 4.7, together with the parameters specified in the algorithm, is described above. The key mechanism in the process is the concatenation of \((N-1)\)-variable boolean functions to form \(N\)-variable boolean functions. This allows a degree of control over the positions and magnitude of Walsh Hadamard transform values being added together at each concatenation level. For example, corresponding Walsh Hadamard transform entries in position \(\omega\) of two \((N-1)\)-variable boolean functions, having equal magnitude and opposite sign, when concatenated will provide a zero Walsh Hadamard transform value in position \(\omega\) of the resulting \(N\)-variable function. This may assist the achievement of an increased order of correlation immunity if \(hw(\omega) \leq m\) and \(\hat{F}(\omega) = 0\) for all other \(\omega\) where \(hw(\omega) \leq m\). Further, the effect of higher Walsh Hadamard transform values corresponding to lower Walsh Hadamard transform values when concatenated will reduce the total entry in the \(N\)-variable boolean function and consequently increase the nonlinearity of the function if other values in the vector do not exceed this. We apply Method 2 to focus on the properties of balance, high nonlinearity and correlation immunity as each property may be derived and computed from the Walsh Hadamard transform of a boolean function, and the concatenation process is highly suitable for the direct manipulation of Walsh Hadamard transform values in order to optimize these properties.

Experimental Results

The main focus of experiments using Method 2 have involved the generation of highly nonlinear resilient \(N\)-variable boolean functions \((5 \leq N \leq 9)\) with algebraic degree maximizing Siegenthaler’s inequality (see Theorem 2.5). Trials for all possible orders of resilience were conducted for each \(N\) in this range. Table 4.5 contains the best combinations of property values exhibited by functions generated by Method 2. We express these results in \((N,m,\text{deg},NL)\) notation, where \(N\) is the dimension, \(m\) is the order of resilience, \(\text{deg}\) is the algebraic degree and \(NL\) represents the nonlinearity of the function. Note that, in subsequent discussions, we also utilize \((N,m,NL)\) notation.

The combined properties in the table exhibited by boolean functions generated
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

Table 4.5: Optimal combinations of property values known at the time of this research, which were able to be achieved by Method 2. Note, however, that (8,1,6,112) is not optimal for nonlinearity.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Optimal known functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>(5,1,3,12), (5,2,2,8)</td>
</tr>
<tr>
<td>6</td>
<td>(6,1,4,24), (6,2,3,24), (6,3,2,16)</td>
</tr>
<tr>
<td>7</td>
<td>(7,1,5,56), (7,2,4,56), (7,3,3,48), (7,4,2,32)</td>
</tr>
<tr>
<td>8</td>
<td>(8,1,6,112), (8,2,5,112), (8,3,4,112), (8,4,3,96), (8,5,2,64)</td>
</tr>
<tr>
<td>9</td>
<td>(9,1,7,240), (9,2,5,240), (9,3,5,224), (9,4,4,192), (9,5,3,192), (9,6,2,128)</td>
</tr>
</tbody>
</table>

by Method 2 were all able to maximize Siegenthaler’s inequality with respect to the number of input variables, order of correlation immunity and algebraic degree, as well as achieve nearly all optimal nonlinearity values known. Note that (9,2,6,240) functions have not yet been found by any method.

The set sizes for the starting pool of 4-variable functions are dependent on the number of functions which satisfy the selection criteria specified at that level. As the selection criteria can be varied for each program execution, these 4-variable set sizes will differ accordingly. For our experimental runs, typical set sizes for the lower dimensional functions (eg $N = 5$ and 6) were between 30,000 and 50,000. Set sizes for the higher dimensional functions ($N = 7$, 8 and 9) were typically between 1,000 to 10,000 but usually tending to 1,000 for the target $N$-variable set. It is, however, a flexible parameter of Method 2 and may be decreased or increased beyond these values to achieve a balance between a desired number of final functions and the execution time.

Example truth tables (in hex notation) of some of the best known functions that have been obtained by Method 2 are listed below. We also set out below tables containing the typical parameters at each level of the concatenation process, $N = 4$ to target $N$, which were used to achieve these results. The reader should note, however, that most of the property combinations are able to be achieved by a number of different sets of parameters. In the table, $WHT_{max}^N$ is the maximum Walsh Hadamard transform vector value and $CI_N$ is the minimum order of correlation immunity (actually resilience) imposed at each level $N$.

A 7-variable $CI(2)$ balanced boolean function with a maximal nonlinearity of 56 and algebraic degree 4 is presented in Example 4.8. One way to achieve functions with these properties was to firstly concatenate 4-variable $CI(0)$ functions with nonlinearity at least 4 to achieve a list of (5,1,12) functions. The
Example 4.8: (7,2,4,56)

6369d82d56ac8b71499bb5c27a64863d

<table>
<thead>
<tr>
<th>N</th>
<th>$WHT_{max,N}$</th>
<th>$CI_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

elements of a set of these 5-variable functions are then concatenated to achieve a list of 2-resilient 6-variable functions with nonlinearity at least 24. Finally, the elements of a set of these 6-variable functions are concatenated to achieve the target 7-variable functions which are 2-resilient with nonlinearity 56.

Example 4.9: (8,1,6,112)

54aa27d9eb324c56229ac4fe5341c813dc2f69f29f8054d978292cb2e356

<table>
<thead>
<tr>
<th>N</th>
<th>$WHT_{max,N}$</th>
<th>$CI_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

An 8-variable 1-resilient boolean function with optimal algebraic degree, and nonlinearity of 112 is presented in Example 4.9. One set of Method 2 parameters which achieves many functions with these properties is listed in the table above. A specified number of 4-variable $CI(0)$ functions with nonlinearity at least 4 are concatenated to form a list of (5,0,12) functions. These in turn are concatenated to form possible (6,1,24) functions. A number of 1-resilient 7-variable functions with a minimum nonlinearity of 48 are stored after combining (6,1,24) functions, and finally target (8,1,6,112) functions are obtained from the concatenation of the stored list of 7-variable functions.

A sample (9,2,5,240) boolean function is presented in Example 4.10, together with one possible set of Method 2 parameters for generating such functions. For these parameters, 4-variable $CI(0)$ functions with nonlinearity at least 4 are con-
Example 4.10: (9,2,5,240)

92cd3d629e31c16ea51d96b8d2a64b5692cd3d629e31c16ea51d96b8d2a64b56
d83c2769c32796d849b567a8ba49586727c3d8963cd86927b64a985745b6a798

catenated to form a list of (5,1,12) functions. The concatenation process continues with the (5,1,12) functions combining to achieve (6,2,24) functions. Then, a list of (7,2,56) functions with a three-valued Walsh Hadamard spectra are stored from the concatenated (6,2,24) functions. These (7,2,56) functions are concatenated and a list of 8-variable 2-resilient functions are stored with nonlinearity 112. Finally, the target functions, (9,2,5,240), are achieved by combining the stored 8-variable functions. This example illustrates that it is sometimes useful to concatenate plateaued functions (see Section 2.1.4) in order to obtain higher dimension functions with good measures of these combined properties.

A discussion on comparisons between the results of Method 2 and those of other heuristic techniques and algebraic constructions will now follow.

In [67], the following nonlinearity results were produced for 1-resilient boolean functions using a Genetic Algorithm:

<table>
<thead>
<tr>
<th>N</th>
<th>WHTmaxN</th>
<th>WHTminN</th>
<th>CI_N</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.6: Best nonlinearity achieved by Genetic Algorithm in [67] for 1-resilient boolean functions. A dash in the table indicates that no algebraic degree was reported.

When comparing the best 8 and 9-variable function nonlinearity values achieve-
able in Table 4.6 with those generated by Method 2, it is clear that, for these input values, Method 2 has greatly improved on these results (particularly in terms of the nonlinearity of 9-variable resilient functions) as well as being flexible enough to achieve varying orders of resilience and algebraic degrees maximizing Siegenthaler’s inequality.

A heuristic method referred to as the Directed Search Algorithm involving selected single bit complementation in the truth table of random balanced boolean functions, similar to hill climbing, was presented in [78]. This method was designed to generate highly nonlinear 1-resilient boolean functions, relying on a linear transformation to achieve resilience. Their best results are recorded in Table 4.7.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$(N, m, \text{deg}, NL)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>(8,1,6,112)</td>
</tr>
<tr>
<td>9</td>
<td>(9,1,7,232)</td>
</tr>
<tr>
<td>10</td>
<td>(10,1,8,476)</td>
</tr>
<tr>
<td>11</td>
<td>(11,1,9,976)</td>
</tr>
<tr>
<td>12</td>
<td>(12,1,10,1972)</td>
</tr>
</tbody>
</table>

Table 4.7: Best nonlinearity achieved by Directed Search Algorithm in [78] for 1-resilient boolean functions and also obtaining highest algebraic degree.

For 8 and 9-variable boolean functions Method 2 is able to easily generate equal or better results. More specifically, Method 2 can achieve nonlinearities of 240, an improvement over the Directed Search Algorithm’s best nonlinearity of 232, for 9-variable 1-resilient functions whilst maximizing Siegenthaler’s inequality. The authors of [78] report taking up to two weeks to generate their results. By comparison, Method 2 runs have taken a few hours at most for $N = 9$ and up to a few minutes for $N = 8$ to produce 1000 functions each with these or better property values.

Two resilient function construction techniques were proposed in [85] which were referred to as Algorithm A and Algorithm B. The main construction step in Algorithm A is the concatenation of a sequence of functions made up of direct sums of strings with maximum possible nonlinearity, and linear functions. Algorithm B is based on the concatenation of one nonlinear resilient $(m+2)$-variable function with multiple linear functions similar to Algorithm A. Their results using these construction techniques are shown in Table 4.8 in $(N,m,NL)$ notation.
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

<table>
<thead>
<tr>
<th>$N$</th>
<th>Algorithm A</th>
<th>Algorithm B</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>(8,1,112), (8,3,96), (8,4,96)</td>
<td>(8,2,112), (8,3,80), (8,4,32)</td>
</tr>
<tr>
<td>9</td>
<td>(9,3,224), (9,4,192),</td>
<td>(9,2,232), (9,3,208), (9,4,160)</td>
</tr>
<tr>
<td>10</td>
<td>(10,3,448), (10,4,448)</td>
<td>(10,3,464), (10,4,416)</td>
</tr>
<tr>
<td>11</td>
<td>(11,3,960), (11,4,896)</td>
<td>(11,2,984), (11,3,944), (11,4,928)</td>
</tr>
<tr>
<td>12</td>
<td>(12,3,1920), (12,4,1920)</td>
<td>(12,3,1968), (12,4,1888)</td>
</tr>
</tbody>
</table>

Table 4.8: Best $(N, m, NL)$ property value combinations $(8 \leq N \leq 12)$ constructed by Algorithm A and Algorithm B in [85]

We compare Method 2 with the 8 and 9-variable results in Table 4.8. Neither Algorithm A nor Algorithm B has achieved all possible varying orders of resilience for these number of inputs, as has Method 2. The nonlinearity achieved by functions generated by Method 2 is at least equal to that achieved by these Algorithms but, in most cases, is higher. Methods such as this, involving concatenation with linear functions, may produce functions which are cryptographically weak and more susceptible to attack.

The 8 and 9-variable results achieved by Method 2 and other methods, the comparisons of which have been discussed so far, are summarized in Table 4.9.

<table>
<thead>
<tr>
<th>Method</th>
<th>$(N, m, d, NL)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 2</td>
<td>(8,1,6,112)</td>
</tr>
<tr>
<td></td>
<td>(8,2,5,112)</td>
</tr>
<tr>
<td></td>
<td>(8,3,4,112)</td>
</tr>
<tr>
<td></td>
<td>(8,4,3,96)</td>
</tr>
<tr>
<td></td>
<td>(8,5,2,64)</td>
</tr>
<tr>
<td></td>
<td>(9,1,7,240)</td>
</tr>
<tr>
<td></td>
<td>(9,2,5,240)</td>
</tr>
<tr>
<td></td>
<td>(9,3,5,224)</td>
</tr>
<tr>
<td></td>
<td>(9,4,4,192)</td>
</tr>
<tr>
<td></td>
<td>(9,5,4,192)</td>
</tr>
<tr>
<td></td>
<td>(9,6,2,128)</td>
</tr>
<tr>
<td>Millan et al. [67]</td>
<td>(8,1,-,112)</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>(9,1,-,232)</td>
</tr>
<tr>
<td>Pasalic et al. [78]</td>
<td>(8,1,6,112)</td>
</tr>
<tr>
<td>Directed Search Algorithm</td>
<td>(9,1,7,232)</td>
</tr>
<tr>
<td>Sarkar et al. [85]</td>
<td>(8,1,-,112)</td>
</tr>
<tr>
<td>Algorithm A</td>
<td>(8,3,-,96)</td>
</tr>
<tr>
<td></td>
<td>(8,4,-,96)</td>
</tr>
<tr>
<td></td>
<td>(9,2,-,224)</td>
</tr>
<tr>
<td></td>
<td>(9,4,-,192)</td>
</tr>
<tr>
<td>Sarkar et al. [85]</td>
<td>(8,2,-,112)</td>
</tr>
<tr>
<td>Algorithm B</td>
<td>(8,3,-,80)</td>
</tr>
<tr>
<td></td>
<td>(8,4,-,32)</td>
</tr>
<tr>
<td></td>
<td>(9,2,-,232)</td>
</tr>
<tr>
<td></td>
<td>(9,3,-,208)</td>
</tr>
<tr>
<td></td>
<td>(9,4,-,160)</td>
</tr>
</tbody>
</table>

Table 4.9: Comparison of 8 and 9-variable results, achieved by various methods, focussing on resilience

The authors of [77] proved the existence of $(7,2,56)$ resilient boolean functions by a simple construction which begins with four 5-variable functions from the four $N = 5$ equivalence classes with $NL = 12$. Separately, two pairs of these functions were combined whilst running through affine transformations of these functions.
in an attempt to obtain a large list of (6,2,3,24) functions. (7,2,4,56) functions were obtained by combining certain (6,2,3,24) functions, the first of which was reported to have been found after approximately half an hour. Although, at this time, it was known that the maximal nonlinearity for a 7-variable boolean function was 56, this was the first reported work which showed that such functions could be 2-resilient. At the time of this publication, Method 2 was trialled for (7,2,4,56) resilient boolean functions and found the first of these within seconds on a Pentium III 700 MHz PC. A large number of such functions are able to be generated by Method 2. An example of one set of parameters which may be used by Method 2 for generating (7,2,4,56) resilient boolean functions can be found above in Example 4.8.

Improved results for specific $\langle N, m, d, NL \rangle$ boolean function property combinations were presented in [54]. This solved many open questions about previously unknown combinations of optimal properties and was achieved by new construction methods. In particular, (8,1,6,116), (10,1,8,488), (11,3,7,976) and (12,3,8,1984) functions were able to be constructed and largely represented nonlinearity improvements whilst achieving optimality for order of resilience and algebraic degree.

The construction of (8,1,6,116) functions in [54] involved first constructing unbalanced $[6,1,5,26]$ functions with hamming weight 30. Letting $f$ represent an unbalanced $[6,1,5,26]$ boolean function, then a concatenation of the form $f\|f \|f \|f \|$ may produce an unbalanced function $[8,1,5,116]$ with hamming weight 124 in accordance with Proposition 1 in [54]. The next step in their construction was to complement chosen sets of four zero bits of the truth table in order to achieve balance. The trial ended with the first set of bits which, when complemented, retained the nonlinearity value of 116. The authors then performed a linear transformation in order to achieve the 8-variable 1-resilient function with nonlinearity 116. Finally, it is shown in [54] that if the nonlinearity is 116, as opposed to a value $\leq 112$, then the algebraic degree of this function must be 6. Note that an $(N+2)$-variable function produced in this manner by the concatenation of an $N$-variable function with itself or complement is expected to be cryptographically weak as attacks seeking to decompose such a function seem possible.

As mentioned earlier, Method 2 has only been trialled for $N$-variable functions, $N \in \{5,6,7,8,9\}$, and for the 8-variable case, to date has been unable to generate 1-resilient, maximal algebraic degree functions with a nonlinearity of 116. These
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

limitations will be discussed in the next subsection of this chapter.

The work of [17], provided results of heuristic optimization using simulated annealing to improve several cryptographic properties of boolean functions. Table 4.10 shows a summary of their results for combined $N$, order of resilience, algebraic degree, nonlinearity and autocorrelation (in terms of maximum absolute autocorrelation value).

<table>
<thead>
<tr>
<th>$N$</th>
<th>$(N, m, d, NL, AC_{max})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>(5,1,3,12,8), (5,2,2,8,32)</td>
</tr>
<tr>
<td>6</td>
<td>(6,1,4,24,16), (6,2,3,24,32), (6,3,2,16,64)</td>
</tr>
<tr>
<td>7</td>
<td>(7,1,5,52,32), (7,2,4,56,32), (7,3,3,48,128), (7,4,2,32,128)</td>
</tr>
<tr>
<td>8</td>
<td>(8,1,6,112,40), (8,2,5,112,56), (8,3,3,96,256), (8,4,3,96,256), (8,5,2,64,256)</td>
</tr>
<tr>
<td>9</td>
<td>(9,1,7,232,72), (9,2,6,232,88), (9,5,3,192,512)</td>
</tr>
<tr>
<td>10</td>
<td>(10,1,8,476,104)</td>
</tr>
</tbody>
</table>

Table 4.10: Combination of property values achieved by [17] using simulated annealing where $5 \leq N \leq 10$

Although Method 2 has not considered autocorrelation in its computation, some comparison can be made for all other properties with the results in Table 4.10. For $5 \leq N \leq 9$, Method 2 was able to generate functions with property values at least equal to those obtained by the simulated annealing techniques used in [17] and in some instances with better property values. For example, Method 2 was easily able to generate many $(7,1,5,56)$ functions which have maximum nonlinearity, as well as $(8,3,4,112)$ functions with algebraic degree maximizing Siegenthaler’s inequality. In addition, Method 2 achieves 9-variable resilient functions with nonlinearity 240.

The combined property results of Method 2, as compared with other heuristic methods as well as other approaches, have been discussed. Note that it is very difficult to discuss comparisons between Method 2 computational program runs and other methods due to a lack of this information being provided in other research papers. However, we provide some insight into the computational effort required by Method 2 below.

Method Applicability

The focus of the design of Method 2 was to enable the rapid generation of resilient boolean functions with high nonlinearity. Extensive experiments have proved the
validity of this design for $N$-variable boolean functions where $5 \leq N \leq 9$ in generating many functions with optimal combinations of property values in a computationally efficient manner. The computational effort required for Method 2 ranged from milliseconds for $N = 5$, a few minutes for $N = 8$ and at most a few hours for $N = 9$ in order to generate 1000 functions. Experiments for Method 2 were predominantly conducted using Pentium III computers from 700 MHz to 1000 MHz.

Method 2 is capable of achieving functions with all possible orders of resilience for each $N$ trialled. A clear advantage of Method 2 is that this single method can produce this wide variety of optimal functions. This compares extremely favourably with construction techniques which, although can typically achieve the most optimal combinations of functions, are often restricted to producing functions that are constrained by specific conditions and thus are unable to provide the same broad range of optimal functions. A large number of functions with an algebraic degree maximizing Siegenthaler’s inequality can be easily generated using Method 2. The majority of recent research in the area of resilient functions has had to perform a linear transformation on the resulting boolean function in order to obtain some order of correlation immunity. Method 2 uses concatenation of selected boolean functions to achieve the desired order of correlation immunity and no linear transformations are required. Optimal boolean functions generated using Method 2 are ideal for use in stream cipher design as nonlinear combining functions for linear feedback shift registers.

Experimental research performed for this thesis used starting functions selected from the set of all 4-variable functions and trials in generating target 5 to 9-variable functions were successfully completed. These trials have demonstrated the suitability of this method in generating optimal highly nonlinear resilient boolean functions. An observation of the rate at which the computational times of the above $N$-variable function trials increase indicates significant additional computational effort is required as the target number of variables increases. It is expected that the generation of $N \geq 10$-variable functions would be considerably slower if using 4-variable functions as the starting pool for concatenation. A possible future research direction would involve investigations into the selection of $N \geq 5$-variable functions to commence the concatenation process.

Another limitation of Method 2 is that for larger $\text{target} N$ and small $m$, functions which possess multi-valued Walsh Hadamard transform vector values are in-
creasingly difficult to generate. Regardless, Method 2 has demonstrated the ability to successfully and quickly generate boolean functions with currently known optimal combinations of property values, and a large number of them.

4.3 Boolean Functions Satisfying $PC(k)$ of Order 0

As discussed in the “Avalanche” section of Chapter 2, propagation criteria is a property of boolean functions which provides a measure of the extent to which the change in the output of functions is uniformly distributed, given a change in their input. Thus, propagation criteria allow a determination of how well input changes have spread through a function and affected the output of that function uniformly. A boolean function exhibiting a non-zero degree of propagation criteria, that is $PC(k)$ where $k \geq 0$, is said to have good diffusion by increasing the difficulty of predicting its output given a particular input change. For this reason, such a function is also considered to be less vulnerable to differential cryptanalytic attacks (see Section 2.3.1).

Though this section is devoted to boolean functions satisfying propagation criteria, their importance to the security of cipher systems is somewhat lessened without those same functions possessing a reasonable nonlinearity value. $PC(k)$ functions ($k > 0$) with inadequate nonlinearity values may exhibit resistance to attacks such as differential cryptanalysis but are likely to remain vulnerable to attacks such as linear cryptanalysis. Before discussing a heuristic approach to the generation of boolean functions which consider both of these cryptographic properties, we present a brief summary of work done by others in this area.

4.3.1 Related Work by Other Researchers

Section 4.3.2 of this chapter will focus on obtaining boolean functions satisfying $PC(k)$ using heuristic techniques. However, compared to constructions targeting the properties of resilience and nonlinearity, there is minimal research work that has been reported in the algebraic construction of $PC(k)$ boolean functions. Some of this work will be discussed in Section 4.3.2 of this chapter. For other related, but less relevant, past construction approaches for $PC(k)$ or SAC boolean functions see, for example, [49], [10] and [85]. The reader should also note that re-
cent papers on the construction of multiple output functions (s-boxes) satisfying $PC(k)$ are rare.

4.3.2 Method 3

We now present a third new heuristic method, “Method 3”, for generating boolean functions with a different set of combined properties. The cryptographic properties which are targeted for satisfaction and/or optimization by Method 3 are propagation criteria, high nonlinearity and balance. Method 3 seeks to simultaneously achieve the balance property together with some non-zero degree of propagation criteria, as well as a good measure of nonlinearity for boolean functions with a range of input variables $N$.

Method 3 was developed primarily as a directed search technique for generating boolean functions exhibiting some non-zero degree of propagation criteria of order 0, that is $PC(k)$ (for some $k \in \{1,2,\ldots,N\}$), without fixing subsets of input bits. However, the ease with which the incorporation of balance and nonlinearity criteria was possible was due to the nature of the method. Thus, the algorithm for Method 3 seeks to achieve the simultaneous satisfaction and/or optimization of three desirable cryptographic properties, namely, propagation criteria of degree $k$ ($k > 0$), high nonlinearity and balance.

The basis for being able to simultaneously improve both a function’s degree of propagation criteria and its nonlinearity in this heuristic method arises from the relationship between the autocorrelation function and the Walsh Hadamard transform of $\hat{f}(x)$ as stated in the Wiener-Khintchine Theorem (Theorem 2.7). Specifically, it is possible to determine the values and positions of entries in the autocorrelation function of $\hat{f}(x)$ by performing a summation process on relevant entries in its Walsh Hadamard transform. This relationship can be expressed in the following manner:

$$\hat{r}_f(\alpha)^2 = \sum_{\omega = 0}^{2^N-1} (\hat{F}(\omega)(1 - (hw(\alpha \& \omega) \mod 2)))^2 - \sum_{\omega = 0}^{2^N-1} (\hat{F}(\omega)(hw(\alpha \& \omega) \mod 2))^2$$

$\forall \alpha \in \{0,\ldots,2^N - 1\}$.

Thus, for an $N$-variable boolean function $f(x)$ to satisfy $PC(k)$ ($k > 0$), it becomes necessary to ensure the existence of zero entries in the autocorrelation
function of $\hat{f}(x)$ for all values of $\alpha$ where $1 \leq hw(\alpha) \leq k$. This can be achieved by
the manipulation of values in the Walsh Hadamard transform vector of $\hat{f}(x)$. For
the concurrent improvement of the function’s nonlinearity value, it is sufficient
to consider only the effect on the Walsh Hadamard transform of $\hat{f}(x)$ of selected
bit changes in the corresponding truth table. As stated in Section 2.1.2, the
balance property is satisfied when $\hat{F}(0) = 0$. The algorithm for Method 3 seeks
to collectively optimize the Walsh Hadamard transform distribution such that
the balance property and the best achievable measures of propagation criteria
and nonlinearity are exhibited by the final function.

The execution of Method 3 requires the selection and specification of a number
of parameters which control the operation of the method. These parameters are
as follows:

- $NL_{target_N}$: the desired nonlinearity of the final function
- $PC_{target_N}$: the degree $k$ of propagation criteria (that is, $PC(k)$) desired to
  be exhibited by the final function
- $maxiterations_N$: the maximum number of iterations of the process to be
  performed on any one starting function
- $mindesiredNL_N$: the minimum acceptable nonlinearity for the starting func-
  tion

The first generated $N$-variable boolean function which has a nonlinearity
larger than $mindesiredNL_N$ and a hamming weight which may be within $2^{\left\lfloor \frac{N}{2} \right\rfloor}$
either side of balance, will be accepted as the starting function for Method 3.
Measures which are ascertained at each iteration are the maximum and minimum
Walsh Hadamard transform values of the function (the positions of which form
the improvement sets), the propagation criteria ($PC$) imbalance, and the single
bit bias of the function at that iteration (or the number of ones away from $2^{N-1}$).
Method 3 relies on the fact that $WHT_{\text{max}}$ (as defined in Definition 2.17) may be
reduced by bit complementation in the corresponding position. The choice of bit
or bits in the truth table to complement will also impact on the particular sums
of Walsh Hadamard transform values which need to equate in order to achieve
$PC(k)$.

The criteria which must be satisfied in order for any bit complementation to
be effected is:
4.3. Boolean Functions Satisfying $PC(k)$ of Order 0

- Any bit change must not increase the $PC$ imbalance;
- Any bit change should bring the function closer to balance, but must not increase the initial allowable single bit bias of the starting function;
- Any bit change should result in a lowering of the magnitude of $\hat{F}(\omega) \forall \omega$ in the two improvement sets, $I_1$ and $I_2$ (as defined in Section 3.1.1).

The algorithm for Method 3, utilizing one bit changes in the truth table of $f(x)$, is set out in Algorithm 4.11.

Algorithm 4.11: Method 3

1. Specify $NL_{target_N}, PC_{target_N}, maxiterations_N, mindesiredNL_N$.
2. Generate a starting $N$-variable boolean function $f(x)$ with hamming weight in the range $\{2^{N-1} - 2^{\lceil \frac{N}{2} \rceil}, ..., 2^{N-1} + 2^{\lceil \frac{N}{2} \rceil}\}$ and nonlinearity greater than $mindesiredNL_N$.
3. Iterate:
   
   (a) Calculate improvement sets $I_1$ and $I_2$.
   
   (b) Define the set, $S_1$, of bit positions which, if complemented, result in a reduction in the maximum absolute value of $\hat{F}(\omega) \forall \omega$ in $I_1$ and $I_2$.
   
   (c) Define the set, $S_2$, of bit positions which, if complemented, do not increase the current $PC(k)$ imbalance.
   
   (d) Define the set, $S_3$, of bit positions which, if complemented, do not increase the allowable single bit bias of the starting function.
   
   (e) If $S_1 \cap S_2 \cap S_3 \neq \emptyset$, then $S = S_1 \cap S_2 \cap S_3$, else $S = S_2 \cap S_3$.
   
   (f) If $S_2 \cap S_3 = \emptyset$, go to Step 2.
   
   (g) Degree setting: optional.
   
   (h) Select a bit in position $t \in S$ and let $f(t) = f(t) \oplus 1$.
   
   (i) If $NL_{target_N}$ and $PC_{target_N}$ are met and $f(x)$ is balanced, exit the program.
   
   (j) If $maxiterations_N$ are reached, go to Step 2, else go to Step 3.

Search based methods typically rely on the use of a starting function from where the search process begins. Two schemes for rapidly generating appropriate
starting functions were developed for this thesis, and were applied to Method 3. Depending on the selected nonlinearity of the starting function for a particular trial, the initial functions required for the Method 3 process were either generated randomly, or by one of the two new starting function generation schemes. These two schemes are now described.

Starting Function Generation Scheme A

The first scheme is used to produce an $N$-variable starting function with a minimum nonlinearity value. It may also be used to simultaneously restrict the hamming weight required by the starting function. Recall that the nonlinearity of an $N$-variable boolean function (see Definition 2.17) is defined to be the minimum hamming distance to the set of all $N$-variable affine functions. This first scheme uses the composition of the matrix comprising the set of $N$-variable linear functions and their combinations (linear matrix) to build an $N$-variable boolean function from smaller equal-sized blocks. Let the number of these blocks be $\beta$. The number of bits in each block, $y$, is required to be a power of 2. Each individual block is filled in turn, ensuring that the hamming distance of each block from the corresponding subset of the matrix is controlled. The minimum cumulative hamming distance, once half the blocks have been filled, is also considered. When all blocks are filled, the minimum cumulative hamming distance (which equates to the nonlinearity of the function), and the hamming weight of the filled blocks, are checked to ensure compliance with limits specified by the parameters to the scheme. A process of backtracking and iteration occurs until the desired values are achieved.

This first scheme requires the following parameters:

- $minblockhd$: the minimum acceptable hamming distance between a function and the set of all affine functions for each block.
- $halfhd$: the minimum cumulative hamming distance at the point where half the blocks have been filled.
- $finalNL$: the minimum acceptable nonlinearity for the completed function.
- $maxbias$: the maximum acceptable single bit bias for the completed function.
- $maxtries$: the maximum number of random functions to try per block.

The algorithm for this scheme is presented in Algorithm 4.12.

This scheme was found to be capable of generating starting functions with good nonlinearity very rapidly and in a much more efficient manner than random
Algorithm 4.12: Starting Function Generation Scheme A

1. Specify $\text{minblockhd, halfhd, finalNL, maxbias, maxtries}$.
2. Choose a blocksize $y$ where $y^\frac{1}{4} \in \mathbb{Z}^+$ and $y \mid 2^N$, and $\beta = \frac{2^N}{y}$.
3. Generate the linear matrix, $L_h$, where $h = y^\frac{1}{2}$.
4. Generate the linear matrix, $L_p$, where $p = \beta^\frac{1}{4}$.
5. For all $\beta$ blocks, iterate:
   (a) Generate a random $h$-variable function, $f_b(x)$, as a candidate for block $b$, with $b \in \{1, \ldots, \beta\}$.
   (b) Calculate the set of hamming distances $\text{hd}(f, l)$ between the function $f_b(x)$ and all functions $l_\omega(x)$ of $L_h$, and the set of hamming distances $\text{hd}(f, \overline{l})$ between the function $f_b(x)$ and all functions $\overline{l_\omega(x)}$ of $L_h$.
   (c) If $\text{min}(\text{hd}(f, l), \text{hd}(f, \overline{l})) < \text{minblockhd}$, reject $f_b(x)$.
   (d) If half the blocks have been filled, and the progressive cumulative hamming distance, calculated by summing the relevant entries of $\text{hd}(f, l)$ and $\text{hd}(f, \overline{l})$ as indicated by $L_p$, exceeds $\text{halfhd}$, reject $f_b(x)$.
   (e) If all the blocks have been filled, and the progressive cumulative hamming distance, calculated by summing the relevant entries of $\text{hd}(f, l)$ and $\text{hd}(f, \overline{l})$ as indicated by $L_p$, exceeds $\text{finalNL}$, or $\text{maxbias}$ has been breached, reject $f_b(x)$.
   (f) If $f_b(x)$ is acceptable, then store it, else
      i. If $f_b(x)$ is to be rejected and the number of iterations does not exceed $\text{maxtries}$, then go to Step 5(a) for the current block;
      ii. If the number of iterations exceeds $\text{maxtries}$, discard $f_b(x)$ and the previous block, and go to Step 5(a) for the previous block.

Starting Function Generation Scheme B

A second scheme developed for generating $N$-variable starting functions, required to have a minimum nonlinearity value and hamming weight, commences with the generation of a random bent function of $\eta$ variables ($\eta > N$). This function is then split into two $(\eta - 1)$-variable boolean functions, each of $2^{\eta-1}$
bits. Each of these functions is then split into two more smaller functions. This process is repeated until functions of the desired dimension, $N$, are obtained.

Clearly, this process will repeat a total of $\eta - N$ times, with $2^{\eta-N}$ functions being generated. Each of these final functions is then evaluated for compliance with the desired properties.

The following parameters are required in this second scheme:

- $minNL$: the minimum acceptable nonlinearity for the final function.
- $maxbias$: the maximum acceptable single bit bias for the final function.

The algorithm for the second starting function generation scheme is shown as Algorithm 4.13.

**Algorithm 4.13: Starting Function Generation Scheme B**

1. Specify $minNL$, $maxbias$.
2. Generate an $\eta$-dimensional bent boolean function, $f_{\eta,0}(x)$, $(x = 0, \ldots, 2^\eta - 1)$.
3. Let $i = \eta$.
4. Iterate:
   (a) Split each of the $f_{i,j}(x)$ functions $(j = 0, \ldots, 2^{\eta-i} - 1)$ equally, to form two $(i-1)$-dimensional functions and store these two functions as $f_{i-1,k}$ ($k = 2j$ and $k = 2j + 1$).
   (b) If $i > N$, let $i = i - 1$ and go to Step 4.
5. Check the nonlinearity and hamming weight of all $2^{\eta-N}$ functions, $f_{N,j}(x)$, $(x = 0, \ldots, 2^N - 1$ and $j = 0, \ldots, 2^{\eta-N} - 1$) and discard any that breach $minNL$ or $maxbias$.
6. Randomly select a starting function from the remaining functions.
7. If no functions remain, go to Step 2.

When reasonably high nonlinearity values are required to be exhibited by starting functions, this scheme provides a rapid mechanism for obtaining such functions, particularly compared to random generation.

**Degree Setting**

An optional variation to Method 3 is to attempt to optimize the algebraic
degree of the final function which is output from the process. For every bit in the truth table which could potentially be complemented, the exact ANF terms which will be xored with the existing function are known. Thus, degree setting involves choosing a bit or bits in the set $S$ (from Algorithm 4.11) which forces the addition modulo 2 of terms with the function which include one or more terms of a certain algebraic degree. If this variation is required, then it is invoked in the algorithm only once the desired target degree of propagation criteria has been achieved. Note that degree setting is not a strictly enforced requirement in that, when it is desired to be invoked, it is attempted but successful only if the appropriate bit or bits are available to be complemented.

Contrasted with Method 1’s nonlinearity focus and Method 2’s combined correlation immunity and nonlinearity focus, Method 3 provides a means of targeting improvements to the degree of propagation criteria and nonlinearity of cryptographic functions using a heuristic approach which relies on the relationship between both the autocorrelation function and Walsh Hadamard transform of a function, as well as the relationship between the truth table and the Walsh Hadamard transform of a function. Because bit complementation is not restricted to complementing a pair of distinct elements in the truth table as in Algorithm 3.1 for hill climbing, the approach of Method 3 enables a more diverse search of the space to be explored.

Method 3 has been applied, in this thesis, to boolean functions in order to achieve some non-zero degree of propagation criteria and balance, and also to simultaneously improve their nonlinearity. The results of the computational trials in applying Method 3 are reported in the Experimental Results section of this chapter. Therein, we also discuss the computational effort required to achieve the reported results.

**Experimental Rationale**

The algorithm for Method 3, Algorithm 4.11, is presented above. The primary search objective of Method 3 is to progressively move around the boolean function space in a manner such that each successive iteration of the search reduces the propagation criteria imbalance and eventually finds a function or functions with the target non-zero degree of propagation criteria. As balance and high nonlinearity are considered to be amongst the most essential fundamental properties which should be possessed by cryptographic functions in terms of their secu-
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

rity, Method 3 seeks to additionally achieve balance and optimize nonlinearity. Together, these cryptographic properties (more specifically propagation criteria and nonlinearity) complement each other to produce boolean functions that are suitable for cryptographic applications that require functions with good diffusion properties, whilst being difficult to approximate with linear or affine functions. Note that in the experiments conducted for this thesis using Method 3, other cryptographic properties such as the $AC_{\text{max}}$ and the sum-of-square indicator, as defined in Corollary 2.3, and Corollary 2.4 respectively, are not optimized in any way in the Method 3 process.

Unlike Method 1, where a random bent function was used as a starting function to commence the search with maximum nonlinearity and zero autocorrelation, Method 3 starts with a random function that meets specified minimum nonlinearity and maximum single bit bias conditions. This enables a broader area of the boolean function space to be searched, and also permits the application of Method 3 to odd-dimensional boolean functions. The starting functions used in Method 3 were either generated randomly or by one of the two Starting Function Generation Schemes proposed in Section 4.3.2.

With every iteration, the bit or bits to be changed will ideally, if complemented, result simultaneously in a reduction of the magnitude of the largest Walsh Hadamard transform value, a reduction in the single bit bias of the function, and a reduction of the propagation criteria imbalance. Reducing the propagation criteria imbalance enables the search to move closer towards functions that satisfy the target non-zero degree of propagation criteria. If no bits that achieve all three requirements can be found, then bit changes that increase $WHT_{\text{max}}$ (as defined in Definition 2.17) but favour the other two requirements, are accepted. Thus, within each iteration, priority is given towards satisfying the propagation criteria and balance requirements, with optimizing nonlinearity as a secondary objective.

**Experimental Results**

Computer trials for generating balanced $N$-variable $PC(k)$ boolean functions with high nonlinearity ($N \in \{4,5,..,10\}$) were conducted using Method 3. In the majority of the trials, a balanced function exhibiting first degree propagation criteria, $PC(1)$, was sought. Trials were run to determine the maximum nonlinearity for which this was achievable. As each balanced function satisfying the propagation criteria requirements was discovered, a record of the achieved
4.3. Boolean Functions Satisfying $PC(k)$ of Order 0

nonlinearity, the algebraic degree, the maximum absolute autocorrelation value and the sum-of-square indicator for the best functions was stored.

Table 4.11 shows the property measures of the best boolean functions discovered using Method 3. We express these results in $(N, k, \text{deg}, NL, AC_{\text{max}}, \sigma)$ notation, where $N$ is the dimension, $k$ is the degree of propagation criteria, $\text{deg}$ is the algebraic degree, $NL$ represents the nonlinearity of the function, $AC_{\text{max}}$ is the maximum absolute autocorrelation value, and $\sigma$ represents the sum-of-square indicator. Although not frequently required, degree setting was occasionally invoked in the Method 3 program runs. For example, from Table 4.11 it was used to generate only the $(7,1,6,54,24,32384)$ and $(8,1,7,114,24,93184)$ functions listed in the table to achieve an optimal algebraic degree of $N-1$. Despite the expected increase in computation time, this variation did not seem to affect the order of magnitude of average program run times.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$(N, k, d, NL, AC_{\text{max}}, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$(4,1,3,4,8,640), (4,2,3:4,2,6,4,256)$</td>
</tr>
<tr>
<td>5</td>
<td>$(5,1,4,12,16,1664), (5,2,3,12,16,2048), (5,2,4,10,8,2048), (5,3,2,12,32,2048), (5,4,2,12,32,2048)$</td>
</tr>
<tr>
<td>6</td>
<td>$(6,1,5,26,16,6784), (6,2,5,26,16,6784), (6,4:5:6,3,28,8,4096)$</td>
</tr>
<tr>
<td>7</td>
<td>$(7,1,5,52,32,35840), (7,1,6,54,24,32384), (7,1,5,56,24,26400), (7,2,5,56,24,26400)$</td>
</tr>
<tr>
<td>8</td>
<td>$(8,1,7,112,48,152320), (8,1,7,114,24,93184), (8,1,6,116,24,93952)$</td>
</tr>
<tr>
<td>9</td>
<td>$(9,1,8,232,80,593792)$</td>
</tr>
<tr>
<td>10</td>
<td>$(10,1,9,478,120,2482432), (10,1,8,128,128,2308480)$</td>
</tr>
</tbody>
</table>

Table 4.11: Best measures of property value combinations exhibited by boolean functions generated by Method 3 for $N \in \{4,5,\ldots,10\}$. Note that an * indicates boolean functions which are bent and therefore unbalanced. The absence of an * indicates that balanced boolean functions with these combined property values were found.

Balanced boolean functions with these characteristics of high nonlinearity and a non-zero degree of propagation criteria are highly useful in cryptology in that they represent functions that exhibit a higher resistance to differential cryptanalysis. The success of Method 3 in generating boolean functions exhibiting these three cryptographic properties is highly dependent on a careful selection of parameters for each computational trial. The required parameters are defined in Section 4.3.2, together with a description of the Method 3 algorithm for one bit changes. The more iterations permitted, the wider the search through the
space, as a one bit change to the truth table of a boolean function occurs at each iteration. However, permitting too many iterations results in inefficiencies, as computational effort is wasted searching the neighbourhood of a boolean function that is unlikely to converge to a better function. The number of iterations for a one bit change in the truth table of a function must be at least \(2^{\frac{1}{2}}\), which is the maximum single bit bias accepted in a starting function. This requirement arises because a single bit change per iteration means that in order to achieve balance, it is necessary to have at least the number of iterations as the starting function’s single bit bias. However, the actual number of iterations performed is usually in excess of this minimum, as Method 3 permits the single bit bias to temporarily increase in order to more easily satisfy the \(PC(k)\) requirement. Though note that the number of iterations must be large enough in order to increase the possibility of the desired target nonlinearity being reached from the starting nonlinearity.

Table 4.12 shows examples of typical parameter combinations that were used to successfully generate examples of boolean functions with some of the best combination of property measures able to be achieved by Method 3. Note that these parameter combinations are typical examples only, and that functions with similar properties are able to be produced by other different combinations of values for these parameters.

<table>
<thead>
<tr>
<th>Function</th>
<th>(NL_{target}N)</th>
<th>(PC_{target}N)</th>
<th>maxiterations(_N)</th>
<th>mindesiredNL(_N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((4,1,3,4,8))</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>((5,1,3,12,16))</td>
<td>12</td>
<td>1</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>((5,2,3,12,16))</td>
<td>12</td>
<td>2</td>
<td>24</td>
<td>8</td>
</tr>
<tr>
<td>((5,3,2,12,32))</td>
<td>26</td>
<td>1</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>((6,1,5,26,16))</td>
<td>26</td>
<td>1</td>
<td>32</td>
<td>18</td>
</tr>
<tr>
<td>((6,2,5,26,16))</td>
<td>26</td>
<td>1</td>
<td>32</td>
<td>46</td>
</tr>
<tr>
<td>((7,1,5,56,24))</td>
<td>56</td>
<td>1</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>((7,2,5,56,48))</td>
<td>56</td>
<td>2</td>
<td>48</td>
<td>100</td>
</tr>
<tr>
<td>((8,1,7,112,24))</td>
<td>112</td>
<td>1</td>
<td>48</td>
<td>100</td>
</tr>
<tr>
<td>((8,1,7,114,24))</td>
<td>114</td>
<td>1</td>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>((8,1,6,116,24))</td>
<td>116</td>
<td>1</td>
<td>64</td>
<td>100</td>
</tr>
<tr>
<td>((9,1,8,232,88))</td>
<td>232</td>
<td>1</td>
<td>64</td>
<td>190</td>
</tr>
<tr>
<td>((10,1,9,470,136))</td>
<td>470</td>
<td>1</td>
<td>100</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 4.12: Typical parameter values used to achieve the best functions using Method 3

We now present examples of some of the best functions generated by the
Method 3 program runs. For each example, a summary of the property values exhibited by that function is provided in \((N, k, d, NL, AC_{\text{max}}, \sigma)\) notation, as well as the parameters used to generate the example. In addition, the Walsh Hadamard transform and autocorrelation distributions of the example function are given, with each bracketed pair of values representing the absolute value of an entry in the function and the number of its occurrences.

Example 4.14: A \((6,2,5,26,16,6784)\) balanced boolean function in hex notation \(11fa4485e9bcc9f1\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target (NL)</td>
<td>26</td>
</tr>
<tr>
<td>Target (PC)</td>
<td>2</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>32</td>
</tr>
<tr>
<td>Minimum starting (NL)</td>
<td>18</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>((12,16) (8,24) (4,16) (0,8))</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>((64,1) (16,9) (8,6) (0,48))</td>
</tr>
</tbody>
</table>

Example 4.15 shows a 6-variable balanced boolean function with nonlinearity 24 and satisfying \(PC(1)\). This function has a maximum absolute autocorrelation value of 24, sum-of-square indicator of 10240 and algebraic degree 4.

Example 4.15: A \((6,1,4,24,24,10240)\) balanced boolean function in hex notation \(5407d7427c672bac\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target (NL)</td>
<td>24</td>
</tr>
<tr>
<td>Target (PC)</td>
<td>1</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>24</td>
</tr>
<tr>
<td>Minimum starting (NL)</td>
<td>18</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>((16,8) (8,32) (0,24))</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>((64,1) (24,2) (16,16) (8,14) (0,31))</td>
</tr>
</tbody>
</table>

Example 4.15 is a 6-variable balanced boolean function exhibiting optimal nonlinearity 26 and satisfying \(PC(2)\), with a maximum absolute autocorrelation value of 16, sum-of-square indicator of 6784 and algebraic degree 5.

Example 4.15 shows a 6-variable balanced boolean function with nonlinearity 24 and satisfying \(PC(1)\). This function has a maximum absolute autocorrelation value of 24, sum-of-square indicator of 10240 and algebraic degree 4.

A 7-variable balanced boolean function satisfying \(PC(2)\) is shown as Example 4.16. This function has algebraic degree 5 and has a nonlinearity of 56, which is
Example 4.16: A \((7,2,56,48,26624)\) balanced boolean function in hex notation

\[
09444f61a9a9e5417c7cbec4f4f32
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target (NL)</td>
<td>56</td>
</tr>
<tr>
<td>Target (PC)</td>
<td>2</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>48</td>
</tr>
<tr>
<td>Minimum starting (NL)</td>
<td>46</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>(16,48) (8,64) (0,16)</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>(128,1) (48,2) (32,5) (16,2) (0,118)</td>
</tr>
</tbody>
</table>

The highest possible for a balanced 7-variable boolean function. The maximum absolute autocorrelation value is 48 and the sum-of-square indicator is 26624.

Example 4.17: A \((7,1,52,32,35840)\) balanced boolean function in hex notation

\[
575b52a7642f40c8d3ece95d670883cb
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target (NL)</td>
<td>52</td>
</tr>
<tr>
<td>Target (PC)</td>
<td>1</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>32</td>
</tr>
<tr>
<td>Minimum starting (NL)</td>
<td>46</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>(24,6) (16,36) (8,58) (0,28)</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>(128,1) (32,2) (24,12) (16,27) (8,56) (0,30)</td>
</tr>
</tbody>
</table>

Example 4.17 is a balanced 7-variable \(PC(1)\) function with algebraic degree 5. The nonlinearity of this function is 52, and it has an \(AC_{max}\) of 32. The sum-of-square indicator is 35840.

Example 4.18 is a balanced 7-variable \(PC(1)\) function with algebraic degree 3. The nonlinearity of this function is 56, and it has an \(AC_{max}\) of 64. The sum-of-square indicator is 32768. Note that this function is also plateaued.

An 8-variable balanced boolean function with an algebraic degree of 6, and satisfying \(PC(1)\) was generated using Method 3. This example function has a nonlinearity of 116, a maximum absolute autocorrelation value of 24, a sum-of-square indicator of 93952 and is presented in Example 4.19.

Another example of an 8-variable balanced boolean function is shown here as Example 4.20. This function has an algebraic degree of 6, and satisfies \(PC(1)\).
4.3. Boolean Functions Satisfying $PC(k)$ of Order 0

Example 4.18: A $(7,1,3,56,64,32768)$ balanced boolean function in hex notation

060606f96f6f5ca35c5c5ca3353506f9

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target $NL$</td>
<td>56</td>
</tr>
<tr>
<td>Target $PC$</td>
<td>1</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>32</td>
</tr>
<tr>
<td>Minimum starting $NL$</td>
<td>46</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>(16,64) (0,64)</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>(128,1) (64,4) (0,123)</td>
</tr>
</tbody>
</table>

Example 4.19: A $(8,1,6,116,24,93952)$ balanced boolean function in hex notation

ccfecc01ff4c7f32c1f0c30eb0c2f03d99ab9954a898aa6796a4965ba597a528

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target $NL$</td>
<td>116</td>
</tr>
<tr>
<td>Target $PC$</td>
<td>1</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>64</td>
</tr>
<tr>
<td>Minimum starting $NL$</td>
<td>100</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>(24,44) (16,140) (8,68) (0,4)</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>(256,1) (24,9) (16,78) (8,51) (0,117)</td>
</tr>
</tbody>
</table>

Example 4.20: A $(8,1,6,112,48,152320)$ balanced boolean function in hex notation

05c947cef8fc44dd05960bbdc24ebaa40e9488a5af152b983dffcb11966319bc

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target $NL$</td>
<td>112</td>
</tr>
<tr>
<td>Target $PC$</td>
<td>1</td>
</tr>
<tr>
<td>Maximum iterations</td>
<td>48</td>
</tr>
<tr>
<td>Minimum starting $NL$</td>
<td>100</td>
</tr>
<tr>
<td>WHT Distribution:</td>
<td>(32,19) (24,45) (16,58) (8,83) (0,51)</td>
</tr>
<tr>
<td>AC Distribution:</td>
<td>(256,1) (48,2) (40,7) (32,22) (24,42) (16,65) (8,83) (0,33)</td>
</tr>
</tbody>
</table>

This example function has a nonlinearity of 112, a maximum absolute autocorrelation value of 48 and a sum-of-square indicator of 152320.

We now compare our results obtained from Method 3 with other previously reported work. A direct comparison proved to be very limited, due to the scarcity of
comparable published results and, more particularly, examples of $PC(k)$ boolean functions. Whilst some papers on the theoretical construction of $PC(k)$ functions have been published including [92], [49] and [10], the provision of example functions are rare, as are attainable function property values for properties or measures which are not the focus of these constructions but are nevertheless significant to the strength of cryptographic functions.

In [17], results of experiments using the heuristic technique of simulated annealing were reported. Reports of incorporating a cost function designed to target both the propagation criteria and correlation immunity properties were supported by the provision of example $N$-variable balanced boolean functions ($N \in \{6,7,8\}$) exhibiting $PC(1)$ and $CI(1)$. Table 4.13 shows the properties of the best functions discovered in [17]. Whilst Method 3 does not directly optimize correlation immunity, it may be easily determined if a generated function can be transformed into a $CI(1)$ function by linear transformation. It is well known that this evaluation is performed by examining the rank of the matrix comprising the zero-valued positions in the Walsh Hadamard transform vector of the function. If the rank of this matrix is equal to $N$, then there exists a linear transformation that will make the function $CI(1)$. Note, however, that such a transformation may affect the function’s degree of propagation criteria.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$NL$</th>
<th>$AC_{max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>52</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>112</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 4.13: Best nonlinearity and $AC_{max}$ values achieved by $PC(1)CI(1)$ balanced boolean functions reported in [17]

We comment on the $PC(1)CI(1)$ 6-variable function with a nonlinearity of 24 and an $AC_{max}$ of 32 reported in [17] and note that Method 3 was able to surpass this by easily generating the function in Example 4.15 (and many like functions). Our example balanced function satisfies $PC(1)$ with the same nonlinearity, but has a lower $AC_{max}$ value of 24. As the rank of the matrix comprised of the zero-valued $WHT$ positions is $N$, this example function can thus be made $CI(1)$. Subsequently, a linear transformation was easily found which achieves $CI(1)$ whilst maintaining $PC(1)$. An example of the resultant function is shown in Example 4.21.
Example 4.21: A (6,1,4,24,24,10240) balanced CI(1) boolean function in hex notation

3dec05ac2361f59a

The PC(1)CI(1) 7-variable function reported in [17] had a nonlinearity of 52 and an AC\(_{\text{max}}\) of 32, and was in fact neither PC(1) nor CI(1). Numerous 7-variable balanced PC(1) functions with the same or higher nonlinearity were able to be generated by Method 3, an example of which is shown in Example 4.17. However, notwithstanding whether or not the authors of [17] were able to generate 7-variable balanced PC(1)CI(1) functions with nonlinearity 52 and AC\(_{\text{max}}\) 32, Method 3 surpassed this with the generation of balanced PC(1) functions with optimal nonlinearity of 56, an example of which is provided in Example 4.18 and which is also able to be transformed into a CI(1) function. An example of this function, after linear transformation, to achieve CI(1) whilst maintaining PC(1), can be found as Example 4.22.

Example 4.22: A (7,1,3,56,64,32768) balanced CI(1) boolean function in hex notation

2277e1dd2d1187d2dd8888b4bb8787d2

Example 4.20, generated by Method 3, is an 8-variable balanced PC(1) function with nonlinearity of 112, AC\(_{\text{max}}\) of 48 and able to be transformed into a CI(1) function. This function is equal in all reported properties to the best 8-variable function provided in [17].

Table 4.14 summarizes this comparison between the best function results reported in [17] (6, 7 and 8 variables) and those achievable by Method 3 for these inputs. We express these results in \((N, m, k, NL, AC_{\text{max}})\) notation, where \(N\) is the dimension, \(m\) is the order of resilience, \(k\) is the degree of propagation criteria, \(NL\) represents the nonlinearity of the function, and \(AC_{\text{max}}\) is the absolute indicator value.

Again, the reader is referred to [53] wherein their best and only example of a balanced 6-variable boolean function with nonlinearity 26, algebraic degree 5, AC\(_{\text{max}}\) of 32 and sum-of-square indicator of 7552 was provided. This function also satisfied SAC (see Definition 2.21), which is equivalent to PC(1). Their example
<table>
<thead>
<tr>
<th>Method</th>
<th>( (N, m, k, NL, AC_{max}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clark [17]</td>
<td>( (6,1,1,24,32) )</td>
</tr>
<tr>
<td>Method 3</td>
<td>( (6,1^*,1,24,24) )</td>
</tr>
<tr>
<td>Clark [17]</td>
<td>( (7,1,1,52,32) )</td>
</tr>
<tr>
<td>Method 3</td>
<td>( (7,1^*,1,56,64) )</td>
</tr>
<tr>
<td>Clark [17]</td>
<td>( (8,1,1,112,48) )</td>
</tr>
<tr>
<td>Method 3</td>
<td>( (8,1^*,1,112,48) ) ( (8,0,1,116,24) )</td>
</tr>
</tbody>
</table>

Table 4.14: Comparison of 6, 7 and 8-variable results achieved by [17] and Method 3. An * indicates that the functions are able to be transformed into \( CI(1) \) functions.

function was achieved by computer search and began with initial functions found in previously published works to exhibit good characteristics. A linear transformation followed this process to produce the example 6-variable boolean function. Method 3 is easily able to generate multiple balanced 6-variable \( PC(1) \) functions with both lower \( AC_{max} \) values and sum-of-square indicator. Further, Method 3 is able to generate multiple balanced 6-variable \( PC(2) \) boolean functions with the same low \( AC_{max} \) and sum-of-square indicator values as our \( PC(1) \) functions.

Two construction methods for balanced boolean functions satisfying propagation criteria and achieving high nonlinearity were presented in [92], one for each of even and odd-dimensional boolean functions. These methods incorporated bent functions as their initial functions. The construction for \( N \)-variable boolean functions with even \( N \) involved starting with an \( (N-2) \)-variable bent function and substituting each variable \( x_i \) in its ANF with the variable \( x_{i+2} \) \( (i \in \{1, 2, \ldots, N-2\}) \). The resulting function was subsequently xored with \( x_1 \oplus x_2 \) to produce the constructed \( N \)-variable boolean function, \( N \) even. In the odd \( N \) construction, the \( (N-1) \)-variable starting bent function has each input variable \( x_j \) substituted by the variable \( x_{j+1} \) in each term in its ANF. The resulting boolean function is then xored with \( x_1 \) to produce the constructed \( N \)-variable boolean function, \( N \) odd. Each of the two methods construct balanced boolean functions satisfying some non-zero degree of propagation criteria. In addition, their lower bound on nonlinearity was proven.

The results of only one 7-variable and one 12-variable function were reported in [92]. An example of a 7-variable balanced \( PC(2) \) boolean function produced by the odd \( N \) construction was provided in [92]. The nonlinearity of this function was given to be 56. It was determined that the algebraic degree of this function
was 3, $AC_{\text{max}}$ equal to 128 and sum-of-square indicator of 32768. This can be compared to the 7-variable balanced boolean function results of Method 3 which is capable of generating multiple $PC(2)$ functions with a nonlinearity of 56. One of our best example 7-variable $PC(2)$ functions generated by Method 3 possesses a higher algebraic degree of 5, a much lower $AC_{\text{max}}$ of 48 (which value also proves the absence of any non-zero linear structures) and a lower sum-of-square indicator of 26624. Note carefully that the example function of [92] contains a non-zero linear structure, which is highly undesirable. Further, note that their function represents an example illustrating Theorem 2 of [109].

The majority of experimental computer trials using Method 3 to generate $N$-variable boolean functions for $N \leq 7$ were performed on a Pentium III 800 Hz PC, whilst trials for $N > 7$ were mostly performed on a Pentium 4 3 GHz PC. The average program run times taken by the Method 3 program to generate a single balanced boolean function satisfying $PC(1)$ for $N \in \{4,5,6,7,8,9,10\}$ are outlined in Table 4.15. As can be seen from the table, these times range from almost instantaneously for $N \leq 6$ for highest possible nonlinearities, a few minutes or less for $N = 8$ with target nonlinearities of 112, to around 20 minutes for $N = 10$ for nonlinearities of 470. Note that, for functions of dimension $N \geq 8$, finding functions with higher nonlinearities than outlined in the table often took more (but not consistently more) time than the orders of magnitude reported here. Also note that there was some degree of variance in the time required to find similar functions but using other different sets of parameter values, as would be expected given the random nature of heuristic techniques.

<table>
<thead>
<tr>
<th>$N$</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational program run times O(minutes)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Average nonlinearities achieved for these times</td>
<td>4</td>
<td>12</td>
<td>26</td>
<td>56</td>
<td>112</td>
<td>230</td>
<td>470</td>
</tr>
</tbody>
</table>

Table 4.15: Order of magnitude of program run times for Method 3 to achieve $PC(1)$ $N$-variable boolean functions with $4 \leq N \leq 10$ and targeting the given nonlinearity values.

**Method Applicability**

Using Method 3, it was possible to successfully and consistently generate balanced boolean functions that exhibited the target non-zero degree of propagation criteria and an acceptably high nonlinearity for $4 \leq N \leq 10$. The $AC_{\text{max}}$ value and
sum-of-square indicator, whilst not being directly optimized in Method 3, were selectively stored and found to be cryptographically good (that is, a low $AC_{max}$ and a low sum-of-square indicator). Due to the optional degree setting feature of Method 3, it is possible to optimize the algebraic degree of boolean functions generated by this method. Note that since, in general, the algebraic degrees of functions output from the method were consistently high, degree setting was not often required. However, when it was invoked, it was often successful in achieving higher algebraic degrees.

Although there was a noticeable increase in the computational effort required to obtain functions with very high nonlinearities for $N \geq 8$, this is expected since Method 3 prioritizes the requirements of non-zero degree of propagation criteria and balance ahead of nonlinearity requirements within each iteration of the process.

The success in finding $N$-variable balanced, highly nonlinear functions with a non-zero degree of propagation criteria ($N \in \{4,5,6,7,8,9,10\}$) means that Method 3 is a suitable and promising heuristic method that is practical for use in the generation of functions to be utilized in cryptographic applications where resistance to both linear and differential cryptanalysis is important. The non-zero degree of propagation criteria coupled with reasonably high nonlinearity leads to functions that tend to exhibit low values of $AC_{max}$. In addition, experimental trials conducted using Method 3 have not resulted in any functions with non-zero linear structures being observed. The generation of balanced $PC(k)$ boolean functions with both odd and even numbers of inputs further demonstrates the general flexibility of Method 3. Also, the key criteria for the design of Method 3 is such that it enables the algebraic degree to be optimized in parallel (should this be required) with the other three cryptographic properties targeted by this method.

Method 3 is able to generate the desired boolean functions with not only good nonlinearity, but also exhibiting good diffusion characteristics for incorporation into cipher systems to enhance their security. Though a significantly larger amount of computational effort would be required, a logical extension to this method would be the incorporation of conditions into the algorithm which sought to simultaneously achieve some non-zero order of correlation immunity.
4.4 Summary

Three new heuristic methods for the optimization of boolean function properties have been designed and described in this chapter. A brief discussion of other work performed by other researchers in the optimization of the cryptographic properties of interest in this chapter is provided. The application of three new heuristic methods developed for this thesis is extensively discussed. The results presented in this chapter have clearly demonstrated the viability of the new heuristic methods designed for this thesis and have also provided evidence as to the success of these methods in generating boolean functions with some of the best known combinations of properties achieved by heuristics to that date.

The directed search process of the first method sought to arrive at boolean functions possessing high nonlinearity and balance, by commencing with a starting function that was bent and therefore having maximal nonlinearity, zero autocorrelation, but without balance. Thus, this method is applicable to the space of even-dimensional functions. From Corollary 2.1 and a similar principle for the autocorrelation function, the key factor of this method was the expectation that guided searching in the neighbourhood of bent functions (which exhibit maximal and perfect nonlinearity) would enable the efficient generation of highly nonlinear balanced boolean functions with low autocorrelation.

Specifically, Method 1 was applied to $N$-variable boolean functions, $N \in \{6,8,10,12,14\}$, and not only was it found to be extremely successful in generating many examples of balanced boolean functions with very high nonlinearities, as well as low $AC_{max}$ values, optimum algebraic degree, and no non-zero linear structures, but the search algorithm was shown to have excellent performance in terms of computational efficiency. This was experimentally observed through the performing of extensive unreported trials on the same processors using the Hill Climbing Method and Genetic Algorithm (just prior to the s-box research of Chapter 5) in an attempt to generate highly nonlinear boolean functions of the same dimensions. Publications describing other methods such as Simulated Annealing [17] and Directed Search Algorithm [78], though not reporting specific times, indicated the comparatively higher computational effort required to generate their results. The results from Method 1, when compared against other heuristic methods discussed earlier in this chapter that targeted the same properties, showed that Method 1 was at least the equal of, and in many cases much better than, these other heuristic methods in terms of results.
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization

The second new heuristic method described in this chapter is based on the concatenation of selected lower-dimensional boolean functions to build target higher-dimensional \( t \)-resilient boolean functions with high nonlinearity and optimized algebraic degree. For each level in the process, pairs of functions to be concatenated were selectively drawn from a pool of functions that possessed particular characteristics. The main design objective was in the selection process in ensuring that concatenation of a pair of functions resulted in a function that exhibited high nonlinearity and that satisfied the desired order of resilience, based on being able to control the magnitude and distribution of values in the Walsh Hadamard transform vectors of each function in the concatenation pair.

The experiments conducted using Method 2, and reported in this chapter, showed that the successive concatenation of lower dimensional boolean functions to form higher dimensional boolean functions with final functions being \( t \)-resilient and highly nonlinear, with algebraic degree maximizing Siegenthaler’s inequality (see Theorem 2.5), consistently produced exceptional results for varying orders of resilience. The novel idea of directly manipulating the Walsh Hadamard transform values in the concatenating pair of functions was paramount in controlling the magnitude and positions of these values, thus successfully generating boolean functions exhibiting the desired combination of cryptographic properties.

The third new method designed for this thesis used heuristics to obtain boolean functions that were balanced and possessed some non-zero degree of propagation criteria, \( PC(k) \). The design idea instrumental to the likely success of this method was to use intelligent iterative bit manipulation based on the relationship existing between particular representations, transforms and measures of a function (specifically the starting function) in order to generate a function with the desired combination of properties, which included high nonlinearity. Two new generation schemes were proposed and utilized in this method for the starting functions required in the computational process. An option to this method extended the bit manipulation strategy to also force a high algebraic degree, where possible.

Method 3 focussed on generating \( N \)-variable balanced boolean functions that satisfied \( PC(k), k > 0 \), and also exhibited high nonlinearity. Experiments conducted, for \( N \in \{4, \ldots, 10\} \) demonstrated that the effectiveness of Method 3 in generating such functions was at least equal to, and often surpassed, that of the limited comparable results of other methods. In addition, the optional abil-
ity to optimize algebraic degree, and the absence of non-zero linear structures in all resulting functions, showed the flexibility and strengthening qualities of this method. Further, the ability to minimize both $AC_{max}$ and $\sigma$ is inherent in Method 3 due to the manner in which the non-zero degree of propagation criteria is achieved.

The new heuristic methods described in this chapter represent a significant contribution to the understanding of how different heuristic approaches can be used to target a range of desired cryptographic properties. The details and results of the experiments conducted to assess and evaluate the effectiveness of these three new heuristic methods are also presented. These three new heuristic methods each focus primarily on targeting a different important cryptographic property, but each is able to achieve a combination of good property values. Together these methods form part of a boolean function toolbox that is capable of generating multiple strong boolean functions for inclusion into cipher systems.

The following two chapters will now shift the focus of the thesis to the application of heuristic techniques for the cryptographic property optimization of multiple output boolean functions (s-boxes).
Chapter 4. The Development and Application of New Heuristic Methods to Boolean Function Property Optimization
Chapter 5

Application of Heuristic Techniques to Substitution Box Analysis and Property Optimization

Substitution boxes (s-boxes) are typically used in the iterative round functions of block ciphers (for example [89], [2] and [19]), but have also been used as components of keystream generators in stream ciphers (for example [36] and [81]) and in the round function of cryptographic hash functions (for example [3]). In terms of operations, s-boxes are one of the few nonlinear components of cipher systems. They are also capable of providing additional cryptographic properties to a cipher, and add to the complexity of the system as a whole. S-boxes which exhibit a good measure of desirable properties peculiar to their application make a significant contribution to the security of cryptographic cipher systems.

In view of very successful existing cryptanalytic attacks on cipher systems which attempt to exploit weaknesses in cipher components, the analysis and optimization of s-boxes and their properties is an ongoing area of important necessary research. The application of existing heuristic techniques to the generation of $N \times M$ s-boxes in order to optimize certain cryptographic properties is the focus of this chapter. More specifically, this avenue of research is investigated for partitions of s-box sizes, namely the cases where $N \geq M$ and $N < M$. Heuristic
techniques were chosen as the tool for this task as they had already proven to be effective in boolean function property optimization [66], [67], [68] as well as in bijective s-box property optimization [64].

The preceding two chapters of this thesis discussed various new and existing heuristic techniques as applied to single output boolean functions. This chapter extends the application of heuristic techniques from boolean functions to s-boxes and represents a large contribution to the research challenge of improving the cryptographic properties of s-boxes and thus increasing the resistance of ciphers incorporating s-boxes to cryptanalysis.

The first section of this chapter describes the application of a Genetic Algorithm and Hill Climbing Method to the improvement of the cryptographic properties of \( N \times M \) regular s-boxes (\( N > M \)). The results of a series of experiments for this research are presented and discussed. The next section of this chapter details a heuristic approach to the generation of practical \( N \times M \) s-boxes (\( N < M \)) meeting particular specified criteria. A summary of the new research concludes this chapter.

### 5.1 \( N \times M \) Regular S-Box Generation (\( N > M \)) using Genetic Algorithm/Hill Climbing

As has been previously identified, many block ciphers incorporate s-boxes in their iterative round functions. Depending on the particular design and structure of the cipher, the input-output dimensionality \( N \times M \) of the s-box will vary. Ciphers exist which contain \( N \times M \) s-boxes with either \( N = M \) (eg Twofish [88]), \( N > M \) (eg Data Encryption Standard (DES) [74]) or \( N < M \) (eg CAST-256 [1]).

In this section we present the results of new work performed in the area of \( N \times M \) regular s-box generation (\( N > M \)) using the known heuristic techniques of Genetic Algorithm and Hill Climbing. The main purpose of this work was to trial the effectiveness of these heuristics in optimizing the nonlinearity of \( N \times M \) regular s-boxes where the number of input bits was larger than the number of output bits. These may also be referred to as surjective s-boxes (see Section 2.2.1).
5.1. Experimental Rationale

Previous work by other researchers [65], [66], [67], [68] have reported on the application of Genetic Algorithms and hill climbing techniques to optimize the cryptographic properties of single output boolean functions. This has been summarized in Section 3.1 of this thesis.

In [64] the Hill Climbing Method was applied to bijective s-boxes for $N = 5$ to 8 inclusive. This heuristic was used to generate bijective s-boxes of this size with a higher nonlinearity than that typically achieved through a random generation process. The hill climbing process involved swapping particular pairs of s-box output entries which resulted in an increase in the nonlinearity of the s-box. A sample size of 10,000 s-boxes was used, with each $N$ considered a separate experiment. The results presented clearly demonstrated the ability of the hill climbing heuristic to consistently achieve much higher nonlinearity values in the s-boxes than were achievable by random generation.

The new research performed for this section is an extension of the work done in [64]. This new research extends their previously reported work of optimizing the nonlinearity property for $N \times M$ s-boxes using a heuristic technique, but rather than considering just s-boxes with $N = M$, we widen the scope to focus on $N \times M$ s-boxes where $N > M$ i.e. a greater number of input bits than output bits. Additionally, rather than only employing the Hill Climbing Method (s-box variation of Algorithm 3.1) to seek this improvement in the nonlinearity we also use a Genetic Algorithm (s-box variation of Algorithm 3.2) independently. We then experiment with variations of the process by combining the Genetic Algorithm with hill climbing to not only improve the nonlinearity property but also the autocorrelation (in terms of maximum absolute autocorrelation value) of the s-boxes. Thus, the goal of this new research was to generate $N \times M$ regular s-boxes with $N > M$ using a Genetic Algorithm, the Hill Climbing Method, and also a combined Genetic Algorithm/Hill Climbing Method, to improve their nonlinearity and autocorrelation properties. Note that a Genetic Algorithm was the primary heuristic technique used in our work. A description of the Genetic Algorithm can be found in Section 3.1.2 of this thesis. The achieved property results were directly compared with the same properties exhibited by randomly generated $N \times M$ s-boxes of the same size.

For our experiments, an initial pool of $P$ parent regular $N \times M$ s-boxes was randomly generated. The fitness function utilized in the algorithm measured
the nonlinearity and maximum absolute autocorrelation values of the s-boxes.

The breeding of each pair of regular s-boxes in the parent pool was performed, producing $\frac{P(P-1)}{2}$ children. The aim of the breeding process at each generation was to attempt to produce s-boxes with an improved fitness measure over that exhibited by the previous generation. The rationale of the process was that each s-box produced by the breeding of two parent s-boxes inherits some of the characteristics of each parent, whilst developing individual characteristics that differ from the parents. A set of $P + \frac{P(P-1)}{2}$ candidate s-boxes was formed by combining parents and children. The fitness of each of the s-boxes in the set was determined and the set ordered by s-boxes with best to worst fitness. The best $P$ s-boxes in the set were retained to become the next generation in the process. The selection process replicates the “survival of the fittest” notion that is reflective of an evolutionary process. This is expected to result in improvements in the properties measured by the fitness function from generation to generation. The breeding and selection process of the algorithm iterates until the satisfaction of the stopping criteria.

A new breeding scheme, breed(S1, S2), was developed for these experiments. Let S1 be parent one and S2 be parent two, each an $N \times M$ regular s-box. Each of the possible $2^M$ elements have a counter, counter$_k$, where $0 \leq k \leq (2^M - 1)$. These are set up to ensure that no more than $2^{N-M}$ copies of each value appear in the child which contains $2^N$ elements.

Two main cases are considered for each element of both parents. Firstly, where the $i^{th}$ element of S1 and S2 are equal and, secondly, the case where the $i^{th}$ element of S1 and S2 are different ($0 \leq i \leq 2^N - 1$). The algorithm for breeding regular $N \times M$ s-boxes, S1 and S2, is described in Algorithm 5.1.

At all stages of the breeding process, after a child takes a value from the possible set of $2^M$ elements, the counter for that value is immediately incremented. In the majority of cases, the genetic breeding rules will be adhered to in that the $i^{th}$ element of the child will take on the value of either one of the parents. However, a mutation operator in this breeding process is activated when the counters for the $i^{th}$ element for both S1 and S2 are full. In that case, then the child must assume a random (mutated) value, and thus develops characteristics which are different from both parents.

In addition to the specific independent Genetic Algorithm process outlined above, independent hill climbing trials were performed to generate highly nonlin-
ear regular $N \times M$ s-boxes ($N > M$). The Hill Climbing Method utilized (s-box variation) is described in Section 3.1.1 of this thesis. Experiments were also conducted using some variations to the Genetic Algorithm which incorporated hill climbing. In particular, two different forms of a combined Genetic Algorithm/Hill Climbing Method were trialled.

The first form of a combined Genetic Algorithm/Hill Climbing Method trialled involved the hill climbing of each of the children produced by the breeding of each possible pair of parent s-boxes in the parent pool (see Section 3.1.3). The set formed by combining these children with the parent pool was then sorted from best to worst fitness. The next generation of the iterated process comprised the best $P$ s-boxes from this set. Note that this form is significantly more computationally intensive than the Genetic Algorithm by itself, as all children at each generation are hill climbed.

The second form of a combined method utilized in this new research performed the Genetic Algorithm to completion before hill climbing the final pool of solution s-boxes. This process used the Genetic Algorithm to obtain regular s-boxes with the best target properties achievable under this method. It then utilized the hill climbing process to improve upon the target property if a local maximum had not yet been reached.

The stopping criteria used in each of the Genetic Algorithm experiments was
a minimum distance measure between s-boxes in the pool. This was appropriate
due to a tendency for Genetic Algorithms to converge towards like solutions as
the number of generations increased. The degree of similarity between pairs of s-
boxes in the pool was evaluated. The hamming distances between all component
output functions were summed for each distinct pair of s-boxes in the pool. The
program stopped when the distance between pairs of s-boxes in the pool reached
a specified value, typically this was no smaller than $4^*M$.

The entire approach to this work, as outlined above, was driven by an aim to
try and find the best possible heuristic technique or technique combination which
was able to generate strong $N \times M$ s-boxes ($N > M$) exhibiting good measures of
nonlinearity and autocorrelation properties. The focus of this work was also to
ascertain the effect of each method and variation and how they compared with
the performance of random generation. An additional consideration was in deter-
mining whether these methods were practical for use, in terms of computational
times, for these s-box sizes.

5.1.2 Experimental Results

We conducted a number of experimental trials for the generation of $N \times M$ regular
s-boxes using heuristic techniques in order to improve their measures of nonlin-
earity and autocorrelation (in terms of maximum absolute autocorrelation value).
Particular testing involved random regular $N \times M$ s-box generation:
(i) using no further improvement methods, and observing nonlinearity and auto-
correlation frequencies;
(ii) using the Genetic Algorithm to improve nonlinearity and autocorrelation val-
ues;
(iii) using the Hill Climbing Method to improve nonlinearity values;
(iv) using the Genetic Algorithm incorporating the hill climbing of every child
output from the breeding process to improve nonlinearity and autocorrelation
values;
(v) using the Genetic Algorithm with final output pool subsequently hill climbed
to improve nonlinearity and autocorrelation values.

Experiments were conducted on regular s-boxes with $N = 8$ input bits and
$M \in \{2,3,4,5,6,7,8\}$ output bits, although we do not report on $M = 8$ due to
the similarity with [64]. Note that this research work concentrated on the gen-
eration of 8x4 regular s-boxes to improve their nonlinearity and autocorrelation
values. This particular size had been chosen in part because of its computational efficiency. We now report on the results of experiments outlined above. All of the experiments involving the Genetic Algorithm used an initial pool size of $P = 10$ regular s-boxes. The sample size for the five types of experiments above was either 10,000 or 100,000 regular s-boxes. The results of (i) were used as a benchmark for comparison of results from (ii), (iii), (iv) and (v) above.

![Figure 5.2: Nonlinearity -v- Frequency, comparing Genetic Algorithm with Random Regular 8x4 s-box generation](image)

The graphs in Figures 5.2 and 5.3 show the frequency of nonlinearity and autocorrelation values respectively for 8x4 regular s-boxes generated randomly versus the Genetic Algorithm applied to random generations of 100,000 s-boxes. The higher nonlinearity results of the Genetic Algorithm application represent a marked improvement over those achieved by s-boxes which have simply been randomly generated. A further outcome from the Genetic Algorithm application is a shift to lower autocorrelation values with an increase in their frequency, as compared to random regular s-box generation.

The results of comparisons between hill climbing applied to random regular s-box generations and purely random regular generations of 8x4 regular s-boxes can be seen in Figure 5.4 for a sample size of 100,000 s-boxes. A comparison between the two processes for these nonlinearity results clearly demonstrates the
Figure 5.3: Autocorrelation -v- Frequency, comparing Genetic Algorithm with Random Regular 8x4 s-box generation

Figure 5.4: Nonlinearity -v- Frequency, comparing Hill Climbing with Random Regular 8x4 s-box generation
ability of the Hill Climbing Method to make significant improvements to this property in terms of higher nonlinearity values and a greater number of them.

Figure 5.5 shows a graph of nonlinearity versus frequency for a sample size of 10,000 regular 8x4 s-boxes, comparing random regular s-box generation with the independent Genetic Algorithm and also the Genetic Algorithm with the final pool hill climbed. It is clear that both the Genetic Algorithm applications to random regular s-box generations produce s-boxes with higher nonlinearity values and more frequent occurrences of them, than those produced by only random generation. The nonlinearity and frequency results of hill climbing the final pool of the Genetic Algorithm represent only a slight improvement over the Genetic Algorithm applied in isolation.

Figure 5.6 illustrates the results for autocorrelation versus frequency for a sample size of 10,000 8x4 regular s-boxes. The graph shows a comparison between random regular s-box generation, Genetic Algorithm applied to randomly generated s-boxes and the Genetic Algorithm with final pool hill climbed. We observe that the autocorrelation values for the Genetic Algorithm applications to randomly generated s-boxes are consistently lower than the autocorrelation
values which a pure random regular s-box generation tends to produce, and the frequency of s-boxes with these lower values in the former method are greater. Hill climbing the final pool of the Genetic Algorithm does provide an improvement in lower autocorrelation values but not a substantial one.

For both Figures 5.5 and 5.6, the effect of hill climbing the final pool of the Genetic Algorithm is minimal in that moderate improvement in these property values is exhibited when compared to the independent Genetic Algorithm. This indicates that the Genetic Algorithm in isolation is capable of approaching, and often reaching, a local maximum and minimum respectively for these properties.

The following four methods are being compared in Figure 5.7: random regular s-box generation, hill climbing applied to random regular s-box generation, the Genetic Algorithm applied to random regular s-box generation and a combination of hill climbing and the Genetic Algorithm involving hill climbing each of the children produced by the Genetic Algorithm breeding process. The values being measured were nonlinearity versus frequency. The sample size for this experiment was 10,000 s-boxes. It can be seen from the graph that all three heuristic techniques produced a greater number of good nonlinearity values than
random regular generation was able to produce. In addition, these values are much higher than those produced by random generation alone. A comparison between the Hill Climbing Method and the two Genetic Algorithms showed that the Genetic Algorithm process produced superior nonlinearity results than the Hill Climbing Method.

Figures 5.8 and 5.9 display the change in the best achievable nonlinearity and autocorrelation values respectively as the number of s-box output bits, \( M \) increases from 1 to 8 inclusive, and \( N = 8 \). These are compared between random regular s-box generations and Genetic Algorithm implementations. As expected, the difficulty of achieving these properties over all linear combinations for increasing \( M \) forces the best values to worsen for any method. Note that this can be observed in an approximately linear fashion from the graphs. The better quality s-box nonlinearity and autocorrelation property values produced by the Genetic Algorithm in general, as opposed to random generation, is noteworthy. For the number of s-box output bits for which random generation and Genetic Algorithm have produced the same best autocorrelation, the Genetic Algorithm produced that best value much more often.

Figure 5.7: Nonlinearity -v- Frequency, comparing Hill Climbing with Genetic Algorithm and Combined Genetic Algorithm/Hill Climbing and Random Regular 8x4 s-box generation
The increase in the maximum attainable nonlinearity and the decrease in the average hamming distance between parents as the number of Genetic Algorithm iterations increases is displayed in Figure 5.10 for 8x4 regular s-boxes. The graph
5.1. \( N \times M \) Regular S-Box Generation \((N > M)\) using Genetic Algorithm/Hill Climbing

shows how the best nonlinearity exhibits a stepwise increase with successive iterations before levelling out to its final achievable value after about the 40th iteration. Note that the nonlinearity increases will always be in steps of two as the s-boxes are balanced. With each iteration, the convergence of the hamming distance between parents can be observed from the graph. As discussed earlier, this is characteristic behaviour for a Genetic Algorithm.

Figure 5.11 depicts, for a sample size of 1,000 8x4 regular s-boxes, the frequency distribution of the number of Genetic Algorithm iterations before the stopping criteria has been reached. The graph shows that convergence typically occurs within 40 iterations.

<table>
<thead>
<tr>
<th>S-Box size</th>
<th>GA Best Nonlinearity</th>
<th>GA Frequency</th>
<th>GA with HC % Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 x 2</td>
<td>110</td>
<td>342</td>
<td>25%</td>
</tr>
<tr>
<td>8 x 3</td>
<td>108</td>
<td>454</td>
<td>21%</td>
</tr>
<tr>
<td>8 x 4</td>
<td>106</td>
<td>1108</td>
<td>15%</td>
</tr>
<tr>
<td>8 x 5</td>
<td>104</td>
<td>2013</td>
<td>19%</td>
</tr>
<tr>
<td>8 x 6</td>
<td>104</td>
<td>9</td>
<td>0%</td>
</tr>
<tr>
<td>8 x 7</td>
<td>102</td>
<td>17</td>
<td>0%</td>
</tr>
</tbody>
</table>
Chapter 5. Application of Heuristic Techniques to Substitution Box Analysis and Property Optimization

Figure 5.11: Number of Iterations of Genetic Algorithm until Stopping Criteria Satisfied

Table 5.1 lists the best nonlinearity achievable by the Genetic Algorithm for 8xM regular s-boxes (M ∈ {2,3,...,7}) from a sample size of 10,000 s-boxes. The frequencies of these best values are listed out of 10,000. The table finally highlights the percentage of s-boxes which were able to have their nonlinearity improved by hill climbing the final pool of 10 s-boxes at the conclusion of the Genetic Algorithm process. These percentages indicate that the Genetic Algorithm was able to achieve a local maximum for nonlinearity for over 75% of cases without needing to apply the Hill Climbing Method on the final pool of s-boxes. Further, as the size of the s-boxes increased, the effect of hill climbing the final pool of s-boxes became negligible, as the effort required to reach a local maxima using the Genetic Algorithm reduced the likelihood that the maxima could be further improved.

Below we outline some of the additional variations tested for the Genetic Algorithm on 8x4 regular s-boxes:

Resetting after every iteration of the Genetic Algorithm: The effect of resetting the Genetic Algorithm after every iteration brought too much randomness into the procedure, causing the results to fluctuate a great deal of the time. This variation could not be trusted to produce consistent results which could be explained. The conclusion drawn was that this was not a good approach to take
and would not contribute to strengthening the s-boxes in a significant way.

Varying the number of iterations of the Genetic Algorithm: The number of iterations which the Genetic Algorithm performed was varied between 20 and 90 (increasing by ten each time). It was found, that for 8x4 s-boxes, 30 iterations consistently seemed to produce the greatest number of good nonlinearity values. 40 iterations were occasionally seen to produce better results and for iterating 50 or more times, the algorithm’s solutions seemed to level off and exhibit no improvement whatsoever as the number of iterations increased.

Resetting in the event that there has been no improvement in results after $x$ iterations: This variation was tested for $x \in \{5, 10, 15, 20\}$. The most frequent number of best results for nonlinearity were obtained by resetting the algorithm when no improvement was observed after 10 iterations, for a set number of 30 iterations of the Genetic Algorithm.

Hill Climbing the initial pool of the Genetic Algorithm: Experimental results using this variation showed that the nonlinearity values were quite poor. As would be expected, by hill climbing the initial parent pool, the nonlinearity values of the hill-climbed s-boxes were already locally maximum and so by subsequently proceeding with the algorithm, we saw little improvement away from those values.

From extensive experimentation which has been conducted during the course of this thesis, it has been observed that heuristic techniques exhibit significant variation in computational times from sample to sample. With this in mind, we briefly provide a coarsely granular summary on execution times for the generation of $8 \times M$ ($M \in \{2, 3, \ldots, 7\}$) regular s-boxes each for an example sample size of 10,000 s-boxes. The Genetic Algorithm applied solely to random s-box generations exhibited times ranging from a few minutes for $8 \times 2$ s-boxes, less than an hour for $8 \times 3$ s-boxes, a couple of hours to several hours for $8 \times 4$ and $8 \times 5$ s-boxes respectively, to around a day or two for $8 \times 6$ and $8 \times 7$ s-boxes respectively. The variation to the Genetic Algorithm which applies hill climbing to the final pool increases the computational effort by roughly 10%. The combined Genetic Algorithm and Hill Climbing Method which hill climbs each of the children produced from the breeding process takes roughly 50% to 300% more time to run than the Genetic Algorithm on its own. These approximate times were measured for execution on an Intel Pentium II 300MHz PC.

For discussions on the application of construction techniques to the generation
of highly nonlinear s-boxes, the reader is referred to, for example, [15] and [87].

5.1.3 Method Applicability

A variety of experiments were reported in the previous section which compare random regular s-box generation with heuristic techniques applied to random s-box generations. The heuristic techniques used were the Genetic Algorithm and Hill Climbing Method described in Section 3.1, and some combination of the two heuristics. For these experimental trials the number of input bits, $N$, to the regular s-boxes was always held constant at 8. Although the majority of experiments concentrated on $M = 4$ s-box output bits, trials were conducted for $M$ in the range $2 \leq M \leq 7$, and in small part for $M = 8$. The goal of this work was to optimize the nonlinearity and autocorrelation properties in order to strengthen regular $N \times M$ s-boxes with $N > M$. We now discuss the degree of effectiveness of these methods in achieving this goal.

It can be observed from experimental results that both the Genetic Algorithm and the Hill Climbing Method performed on random regular $N \times M$ s-box generations ($N > M$) have proved to consistently provide better results for nonlinearity and autocorrelation (in terms of maximum absolute autocorrelation value) than random regular generation alone. For 8x4 regular s-box generations, the best nonlinearity value which we found was 106. Neither random generation nor hill climbing managed to achieve this value. A combined Genetic Algorithm and Hill Climbing Method (involving the hill climbing of each of the children output after breeding the parent pool) is sometimes capable of improving the target properties and producing more s-boxes with good properties but at the cost of a significant increase in computational effort. Similarly, hill climbing the final pool of the Genetic Algorithm output is able to optimize these properties if a local maximum has not already been reached by the Genetic Algorithm. This variation appears to add an acceptable amount of additional computational time to each program run.

In terms of computational effort required, the Hill Climbing Method is the somewhat quicker method when compared to the Genetic Algorithm. As already indicated, the combined Genetic Algorithm and Hill Climbing Method is considerably slower than each method applied independently. For larger values of $N$, the combined method quickly becomes impractical for use in terms of speed, however, the Hill Climbing Method remains the most viable out of these heuristic
techniques.

The success of these heuristic methods, when applied to random regular s-boxes, in generating stronger s-boxes has been demonstrated for these sizes. This work has progressed the ongoing investigation into ways in which more secure s-boxes may be produced for use as components in cryptographic cipher systems.

5.2 An Example of Practical $N \times M$ S-Box Generation ($N < M$)

In the previous section we outlined new work involving extensive experimentation into applying existing heuristic techniques to generate $N \times M$ s-boxes with $N > M$ and showed the effectiveness of this approach. In recent years we have seen an increase in the use of $N \times M$ s-boxes in cipher systems with $N < M$. We now turn our attention to the application of heuristic techniques to the generation of strong $N \times M$ s-boxes with $N < M$. A greater number of output bits to an s-box than input bits provides the opportunity for a subset of distinct entries to be used out of a possible $2^M$ values. In contrast, in the case where $N > M$, there must exist repeat entries in the s-box.

This work further consolidates the application of heuristic techniques as being an effective means of generating strong cipher components. The flexibility of heuristic approaches can be seen by their success on different partitions of s-box sizes: previously published work [64] on bijective s-boxes; surjective s-box property optimization in the previous section of this chapter; and now, in this section, the focus is on $N \times M$ s-box generation for $N < M$. The particulars of this new work involves the utilization of the Hill Climbing Method applied to randomly generated single output balanced boolean functions in order to construct 8x16 and 8x32 s-boxes with specific requirements imposed with regard to properties and relationships between functions.

The new research work summarized in this section was performed in conjunction with Qualcomm Australia who established the requirements for the s-boxes\(^1\). The goal of this research was to use the outcomes of this work for incorporation into their SOBER family of stream ciphers. Therefore, the effectiveness of heuristic techniques in the generation of practical $N \times M$ s-boxes ($N < M$) will be demonstrated as a result of this new work.

\(^1\)This research is reported with the kind permission of Qualcomm Australia.
5.2.1 Desired Characteristics of Qualcomm S-Boxes

The particular cryptographic purpose of s-boxes used in a given cipher system, together with the specific characteristics which are achievable within a reasonable amount of computational time, are among the factors which largely dictate the requirements imposed on the s-boxes.

For the 8x16 and 8x32 s-boxes generated, the following characteristics were deemed to be essential, for their application discussed in Section 5.2.4:

(i) each of the component boolean functions be balanced;

Ensuring that balanced boolean functions comprised the s-boxes was desired in order to eliminate the existence of an exploitable single bit bias in the individual output functions.

(ii) each of the component boolean functions exhibit high nonlinearity;

The task of merely computing the nonlinearity of an 8x32 s-box (usually involving the determination of the maximum absolute value of the Walsh Hadamard transform, for each of the $2^{32} - 1$ linear combinations of the output functions) requires an extraordinary amount of computational effort. Attempts to optimize the nonlinearity of an 8x32 s-box are currently even more infeasible. A more realistic approach was to ensure that each of the component boolean functions of the s-boxes were highly nonlinear. This s-box security factor was to help to contribute to the resistance of the s-boxes against linear cryptanalysis.

(iii) any correlation between the input bits and the first eight component boolean functions should be low;

Minimization of the deviation from $CI(1)$ for these component functions was a desired characteristic because of the intention of the addition of these bits with the input. Thus, reducing the correlation between these output bits and the input bits was logical in order to contribute to resistance against correlation attacks.

(iv) the deviation away from $2^{N-1}$ of the hamming distances between all distinct pairs of component boolean functions should be low.

A strengthening characteristic of the s-boxes was the restriction of the pairwise imbalance between distinct pairs of component functions. In this way we sought to reduce the correlation between pairs of component boolean functions of each s-box.

It was necessary to select an appropriate technique which had a reasonable chance of achieving the above characteristics.
5.2.2 Techniques Used for Generation of 8x16 and 8x32 S-Boxes

The underlying heuristic technique used for generating the 8x16 and 8x32 Qualcomm s-boxes was the Hill Climbing Method (see Section 3.1.1). At each iteration of the process a balanced 8-variable boolean function was randomly generated. This function was put through a strong hill climbing procedure in order to achieve a desired minimum nonlinearity value. The process of finding the first 8 acceptable output boolean functions included a calculation of the maximum absolute Walsh Hadamard transform value in positions, $\omega$, where $hw(\omega) = 1$. If this value was low and within an acceptable deviation from 0 then this individual requirement was satisfied. When at least one boolean function satisfying these requirements had been stored, the hamming distance between the current function under examination and each of the stored functions was determined. A small deviation from $2^{N-1}$ was acceptable as the distance measure between these pairs. This entire process was iterated until either the target number of boolean functions satisfying all requirements had been reached or a specified maximum number of iterations had been reached, whichever occurred first.

The steps involved in this iterative process are summarized in Algorithm 5.12. In this algorithm, $requiredfunctions (\in \{16, 32\})$ refers to the number of 8-variable balanced boolean functions required to be accepted and stored as components of the s-box. $NL_{\min}$ is the minimum acceptable nonlinearity value required to be exhibited by each component boolean function. The parameter, $thresholdCI$, represents the maximum allowable deviation from first order correlation immunity. $Thresholdhd$ is the maximum allowable deviation from balance between distinct pairs of stored functions. $Maxiterations$ is the maximum number of iterations of the process to try in order to achieve $requiredfunctions$.

5.2.3 Experimental Results

Discussions in this section focus on the parameters used in the experimental trials, and the results achieved by this approach.

As balanced boolean functions were randomly generated to begin the computational process, and the balance was maintained by the hill climbing procedure, the balance requirement was easily satisfied. The minimum nonlinearity specified in the code was 108. The strong hill climbing applied to random balanced
Algorithm 5.12: Generation Process for 8x16 and 8x32 S-Boxes

1. Specify required\textit{functions}, \textit{NL}_{\text{min}}, \textit{thresholdCI}, \textit{thresholdhd} and \textit{maxiterations}.

2. Let \( k = 0 \).

3. Generate a random 8-variable balanced boolean function, \( f(x) \).

4. Perform strong hill climbing on \( f(x) \) to produce \( g(x) \).

5. If \( \text{NL}(g) < \text{NL}_{\text{min}} \), reject \( g(x) \) and return to Step 3.

6. If \( k < 8 \) and \( \hat{G}(\omega) > \text{thresholdCI} \ \forall \omega \) where \( hw(\omega) = 1 \), reject \( g(x) \) and return to Step 3.

7. If \( \text{hd}(g, h) > \text{thresholdhd} \) where \( h(x) \) is each previously accepted function, reject \( g(x) \) and return to Step 3.

8. Accept and store \( h(x) = g(x) \) and increment \( k \).

9. If \( k \neq \text{requiredfunctions} \), return to Step 3.

10. If \( \text{maxiterations} \) has been reached, exit.

8-variable functions achieved nonlinearity values for different sets of 16 and 32 component functions of 108, 110 and 112. The code allowed a deviation from \( CI(1) \) for the first 8 acceptable boolean functions to be at most 16. However, values of 8, 12 and 16 were observed in the generated functions. The specified maximum pairwise hamming distance imbalance between distinct output functions was 10. Experiments produced sets of 16 and 32 8-variable functions such that imbalance values of at most 6, 8 and 10 were found between pairs. Constraining the parameters to limits which were more optimal than those mentioned increased the computational effort of the program and did not always manage to achieve the number of functions targeted.

The values achieved by the individual component functions for the best 8x16 and 8x32 s-boxes generated by this process are listed in the following table:

We provide an example s-box from the above table in \textbf{Appendix A}. The 8x32 s-box with all component boolean functions balanced, minimum nonlinearity
Table 5.2: Best s-boxes in terms of values exhibited by component boolean functions represented as $<\text{balanced}, \text{minimum nonlinearity}, \text{maximum deviation from } CI(1) \text{ for first 8}, \text{maximum imbalance between pairs}>$

<table>
<thead>
<tr>
<th>8x16</th>
<th>8x32</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;balanced,108,8,8&gt;</td>
<td>&lt;balanced,108,8,8&gt;</td>
</tr>
<tr>
<td>&lt;balanced,108,12,6&gt;</td>
<td>&lt;balanced,110,16,10&gt;</td>
</tr>
<tr>
<td>&lt;balanced,110,12,10&gt;</td>
<td>&lt;balanced,112,12,10&gt;</td>
</tr>
</tbody>
</table>

achieved by each of the component functions 112, maximum deviation from $CI(1)$ for the first 8 functions 12, and maximum pairwise hamming distance imbalance between all distinct pairs of component functions 10, is the example given in hexadecimal notation.

Additional analytical information is also provided in Appendix A for this example s-box. This information includes:
- the truth table of each of the component boolean functions;
- the maximum absolute Walsh Hadamard transform value for each of the component boolean functions;
- the maximum absolute autocorrelation value for each of the component boolean functions;
- the frequency distribution of absolute Walsh Hadamard transform values for the total of the 32 output functions;
- the frequency distribution of absolute autocorrelation values for the total of the 32 output functions;
- the imbalance in the number of terms in the algebraic normal form away from $2^{N-1}$ for each of the component boolean functions in the s-box;
- the frequency distribution of the maximum absolute Walsh Hadamard transform values for the total of the 32 output functions;
- the frequency distribution of absolute Walsh Hadamard transform values in positions with hamming weight 1 for the total of the 32 output functions. Note that the first 8 output functions are constrained to have values at most 16 in their weight 1 positions, but typically $\in \{8,12,16\}$;
- the frequency distribution of maximum absolute autocorrelation values for the total of the 32 output functions.

The security features offered by the example 8x32 s-box can be gauged by the extraction of key details from the above information. Some of the details
these values provide include the nonlinearity of component boolean functions, complexity of component boolean functions, autocorrelation values exhibited by the component boolean functions, and the degree of deviation from $CI(1)$ of the total 32 component functions (not just the first 8).

From the experimental results obtained and reported here, it is evident that s-boxes satisfying the stated requirements are able to be successfully generated using the approach discussed in Section 5.2.2.

### 5.2.4 Practical Use of S-Boxes

As previously mentioned, the primary goal of this work was to be able to incorporate selected results into the SOBER family of stream ciphers. Significantly, a large portion of the example s-box given in Appendix A was employed in the SOBER-128 stream cipher [36] and in one of the stream ciphers, SOBER-t32, from the t-class of SOBER stream ciphers [80]. Further, a large portion of one of the 8x16 s-boxes produced by this research was incorporated into SOBER-t16, another stream cipher from the t-class of SOBER stream ciphers [80]. In addition, the example s-box given in Appendix A was utilized in the Turing stream cipher [81].

A full description of the SOBER-128 stream cipher specification can be found in [36]. From [36] we note that the 32-bit keystream blocks of SOBER-128 are produced by a combination keystream generator which feeds multiple linear feedback shift registers (LFSR) through a Nonlinear Filter. The Nonlinear Filter uses a variety of operations combined with a number of table look-ups from an 8x32 s-box. Of particular relevance to this research is the 8x32 s-box incorporated in the Nonlinear Filter of SOBER-128.

From [36], the most significant 8 bits of the 8x32 SOBER-128 s-box output is equal to the Skipjack [76] s-box entries xored with all possible ordered 8-bit values $[0,255]$. The least significant 24 bits of the 8x32 SOBER-128 s-box output is identical to the least significant 24 bits of the example s-box given in Appendix A which is one of the best example s-boxes generated as a result of this research work and chosen for inclusion in SOBER-128.

[80] contains a discussion and description of the t-class of SOBER stream ciphers (which include t16 and t32). The combining function used in the Nonlinear Filter of both SOBER-t16 and SOBER-t32 uses an 8x16 and 8x32 s-box respectively. The least significant 8 and 24 bits respectively of these s-boxes were
produced by the results of this new research.

The Turing stream cipher is described in [81]. One of the fixed s-boxes utilized in the Nonlinear Filter of Turing, which the authors refer to as “Qbox”, is the example 8x32 s-box provided as Appendix A, selected by Qualcomm Australia for incorporation into Turing, and produced by this work.

We have demonstrated the applicability of heuristic techniques, in particular, the Hill Climbing Method in this instance, to the generation of s-boxes with a greater number of output bits than input bits (specifically, 8x16 and 8x32) which exhibit properties which make them suitable for incorporation into practical cipher systems.

5.3 Summary

In this chapter we have demonstrated that the heuristic techniques of Genetic Algorithm, Hill Climbing Method and combined Genetic Algorithm/Hill Climbing were able to successfully generate strong $N \times M$ s-boxes $N > M$ with particular experiments focussing on regular s-boxes with 8 inputs and $M \in \{2,3,4,5,6,7\}$. The target cryptographic properties in the generation process were high nonlinearity and low autocorrelation. All results compared very favourably with random regular s-box generation. The success of these methods applied to s-boxes of these dimensions in terms of improved measures of the target properties were able to be ascertained through this research work. Very importantly, the knowledge gained through observing the tradeoff between the quality of the results and the computational times observed in experiments trialled for the two techniques individually as well as the combined method, provided the experience in deciding when best to utilize each technique for like cryptographic applications. This was particularly useful for the subsequent work undertaken which represented an important contribution to the SOBER family of stream ciphers, and discussed in the next paragraph.

A valuable contribution to practical $N \times M$ s-box generation was achieved through applying the Hill Climbing Method in order to generate 8x16 and 8x32 s-boxes, with very specific requirements in terms of cryptographic property measures, aimed at being able to incorporate selected results into the SOBER family of stream ciphers. This heuristic approach was successful in obtaining cryptographically strong s-boxes of these sizes, some of which were chosen by Qual-
comm Australia for utilization in their SOBER-t16, SOBER-t32, SOBER-128 and Turing stream ciphers. This work further demonstrated the effectiveness and practical use of the Hill Climbing Method, and of heuristic techniques in general, in generating strong cipher components. Further, the experience gained through this research in becoming familiar with the computational effort involved in applying this heuristic method to the optimized properties for these s-box sizes, in large part provided the creative influence to perform the research work in Chapter 6.
Chapter 6

Practical Application of Heuristic Techniques to MARS-Like S-Box Generation

A significant part of the work performed for this thesis on s-box analysis and optimization had arisen from a decision to investigate the properties and method of generation of an existing s-box incorporated into a well known block cipher called MARS [39]. An initial opinion was formed that the computational time taken to generate an s-box which exhibited the properties possessed by the MARS s-box was inconsistent with our expectations based on our past experiences in achieving these sorts of s-box properties using heuristic methods, as described in Chapter 5. We held a belief that, at the very least, the MARS linear properties could be achieved using other property conditions that are known to be efficient for the application of heuristic techniques. The question was whether the actual MARS s-box bounds would be attained using this sort of approach and how long this generation process would take, even though we were of the view that, intuitively, such an approach would be considerably more efficient. The research described in this chapter was motivated by the above. This research has led to the efficient generation of similar s-boxes with improved properties using a known heuristic technique described in Chapter 3.

In this chapter we describe an alternative approach to generating MARS s-boxes, which we refer to as MARS-like s-boxes, with better cryptographic proper-
ties and a significantly improved generation time. We present the technique used in the generation process and highlight the level of security offered by MARS-like s-boxes. This work has contributed to the ability to create and utilize s-boxes of this size efficiently for use in cipher systems. Equally importantly, these generated s-boxes exhibit properties which go some way towards resisting differential and linear cryptanalysis.

The first section in this chapter briefly describes the MARS block cipher and the usage of s-boxes within the MARS cipher. A discussion on the particular requirements imposed by the MARS designers on their s-box, as well as the technique used to generate the MARS s-box, is also provided in this first section. The second section discusses the use of heuristic techniques to generate s-boxes satisfying the MARS s-box requirements, MARS-like s-boxes. The MARS-like s-box property requirements are discussed, and experimental results from this research are also reported in the second section. Finally, a third section presents a summary of this chapter.

6.1 The Block Cipher, MARS

MARS is a symmetric block cipher which processes its data in blocks of 128 bits. The shared key may have a user selected key size of between 128 to 448 bits inclusive, increasing in multiples of 32 bits. The MARS algorithm is based on a Type-3 Feistel network structure ([29], [72]). The MARS cipher was a candidate for the Advanced Encryption Standard (AES) [73] and one of the five finalists in the AES selection process. One of the current uses of MARS is in commercially available products, often as part of the cryptographic protection for network security protocols.

The designers of the MARS block cipher have reported their design rationale in [39] which indicates a strong focus on achieving a balance between providing good security and maintaining a computational efficiency which is comparable to other block ciphers of this type. There were three key principles in their design rationale. Perhaps the most significant of these was the notion of using a different design for the outer rounds of the cipher structure than for the inner rounds, to provide added protection for the cipher’s core. A second principle supports a variety of specific machine operations designed to enhance security without compromising efficiency. The final key principle adhered to in the design
of MARS was to ensure ease of analysis of the entire cipher.

6.1.1 General Design of MARS

To aid the reader in the understanding of the role of the MARS s-boxes, we provide a brief summary of the basic structure of the MARS symmetric block cipher.

The input plaintext is partitioned into 128 bit blocks. Each 128-bit plaintext block is encrypted as four 32-bit words within the MARS algorithm. The ciphertext output is also presented as four 32-bit words. MARS permits the size of the key to vary between 128 bits and 448 bits, represented as between 4 and 14 words respectively. An internal key expansion process transforms the initial key into 40 32-bit word subkeys. As this key expansion process is not integral to the work reported in this thesis, we refer the reader to [39] for a full description.

The encryption of each plaintext block of four 32-bit words is processed through three distinct phases described briefly as follows:

The first phase of MARS is called the forward phase. This phase commences with the xor-sum of the plaintext blocks with corresponding blocks of key material. The output of this addition is subsequently used as the input parameter to eight unkeyed Type-3 Feistel rounds. These Feistel rounds incorporate two s-boxes for the substitution of bytes in each one of the four blocks of data in turn. Each single block of data out of the four is chosen in turn to alter the three remaining blocks of data. At that time it is referred to as the source word, whilst the three other blocks of data are referred to as target words in [39]. The rounds of the first phase also comprise manipulation of the four bytes contained in each word by using addition and exclusive-or operations. For three out of four bytes of each data word, 8 bit directional rotation is performed. The purpose of this first phase is to introduce an adequate amount of mixing of the data blocks to fortify the second phase and to protect the key material by establishing a rapid key avalanche effect. The forward phase aims to provide resistance against chosen-plaintext attacks.

The second phase is referred to in the MARS literature as the cryptographic core. In this phase the four output blocks from the first phase are processed during eight Type-3 Feistel rounds. Each of the rounds in the second phase incorporates a keyed component known as the Expansion Function (E-function) which is heavily relied upon for the security of this phase. The first 32-bit data word is input into
the E-function and three data words (left, middle and right) are produced as output. The function uses addition, multiplication and exclusive-or operations together with a single s-box lookup, directional and data dependent rotations. Key material is combined with data words during the E-function process. A further eight Type-3 Feistel rounds are performed with the three output words from the E-function used in reverse order. Being the primary phase for the security of the cipher, the purpose of the cryptographic core is to provide strong resistance to existing cryptanalytic attacks.

The third phase of the algorithm is called the backwards phase. The four output words from the cryptographic core are input into this final phase and processed during eight unkeyed Type-3 Feistel rounds. The operations used in the third phase, and the steps involved in the process, equate to the decryption of the reverse order output words of the forward phase. The concluding step of the third phase is subkey subtraction from the final four data words, resulting in the ciphertext. In a manner similar to that of the first phase, this backwards phase provides mixing and key avalanche. This phase aims to provide resistance to chosen-ciphertext attacks and to fortify the second phase from inverse attacks.

To illustrate the general structure of MARS we extract Figure 1 from [39] to diagrammatically represent the MARS encryption process. It is shown here as Figure 6.1.

### 6.1.2 Usage of S-Boxes in MARS

MARS uses a fixed 9x32 s-box, S, both as a single s-box but also, during certain parts of the computation, as two 8x32 s-boxes, S0 and S1. In each round of the first and third phases of the MARS algorithm, the 32-bit source word undergoes several byte for word substitutions through the two 8x32 s-boxes, S0 and S1. In the first phase, the first and third lowest order bytes of the source word are the inputs into S0, and the second lowest and highest order bytes are the inputs into S1. The output words of the s-boxes are xored and added to target words in a specific combination. In the third phase, the second lowest and highest order bytes of the source word are the inputs into S0 and the first and third lowest order bytes are the inputs into S1. The output words of the s-boxes are xored and subtracted from target words in a specific combination. A full description of the round structures for these phases can be found in [39].

Utilization of the 9x32 s-box takes place only in the E-function of the crypto-
graphic core where a single substitution occurs for each call of the function. Two calls of the E-function are made for each of the 16 rounds in the second phase. The s-box look-up affects only the left word in the E-function. The 9-bit input into the s-box is the low order 9 bits of the combined E-function input word and a key word. The 32-bit output word of the s-box undergoes addition modulo 2 with the right word at two interim steps and a final data-dependent rotation before becoming the left output word of the E-function.

### 6.1.3 MARS S-Box Property Requirements

Fundamental to the design of any strong cipher is a need for the cipher to have the ability to resist existing cryptanalytic attacks, in particular, linear and differential cryptanalysis. A precondition of this resistance is that any s-boxes used in the cipher also exhibit this characteristic. Therefore, the designers of the MARS cipher, in designing their 9x32 s-box, placed particular emphasis on ensuring that their s-box satisfied a number of linear and differential property requirements. We now outline below these property requirements from [39], and discuss their effect on the security of the s-boxes in general.

![Figure 6.1: High-Level Structure of MARS from [39]](image-url)
Differential Property Requirements

1. S does not contain the all zeros word (0x00000000) or the all ones word (0xffffffff).

The exclusion of these words from S is important to prevent a possible weakness in revealing intermediate values which may help any cryptanalytic process. Specifically, if a 9 bit input value corresponded to an all zeros word being output from S, the subsequent xor with the intermediate right word would produce no change in the word. This weakening may propagate to adjacent operations to reveal more information. Similarly, an all ones s-box entry would result in the complement when xored with the intermediate right word in the E-function.

In the first and third phases of the MARS algorithm, the output of S0 and S1 are xored with, as well as added or subtracted modulo $2^{32}$ from, an intermediate target word. Again the effect of the all zeros word for each operation will produce no change in the intermediate target word. An all ones s-box entry will always produce the complement of the target word when xored with it. Addition and subtraction modulo $2^{32}$ of the all ones word with an intermediate target word will result in subtracting one and adding one respectively to the intermediate word. The presence of these entries increases the predictability of intermediate values and the possibility of revealing additional partial information.

2. Every pair of distinct entries in each of the two 8x32 s-boxes, S0 and S1, differs in at least three out of the four bytes. Equivalently, a pair of words from the same 8x32 s-box may have no more than one byte the same, in the same position.

This requirement eliminates the possibility of a zero output difference of two or more bytes in one or both s-boxes. In general, an output of this type (which has low hamming weight), compared to a random output, is likely to result in a higher differential probability in the active s-box. In the case of MARS, the designers discuss a specific potential attack on the forward mixing phase made possible in the absence of this requirement. The attack shows how a zero difference in two bytes of one of the s-boxes can propagate through the rounds of the mixing phase and increase the value of an 8-round differential characteristic probability.
3. The 9x32 s-box, S, does NOT contain two entries $S[i]$ and $S[j]$, $0 \leq i, j \leq 511$, $(i \neq j)$, such that:

(a) $S[i] = S[j]$ (∃ two identical entries in S);
(b) $S[i] = -S[j]$ (∃ entries in S which are complements);
(c) $S[i] = -S[j]$ (∃ entries in S which sum modulo $2^{32}$ to give zero).

This must be a basic requirement in any s-box expecting to achieve minimum security. If S were to have two identical entries, the input difference $\Delta x = i \oplus j$ ($0 \leq i, j \leq 511$ with $i \neq j$) would result in a zero output difference. Moreover, a zero output difference in an s-box would reduce the number of active s-box lookups thereby contributing to differential characteristics with higher probability. Pairs of s-box entries which are complements or negatives, when operated on with exclusive-or or additions respectively, cancel each other out. This may result in exposure of intermediate values and the deduction of partial information. The MARS E-function, which uses the 9x32 s-box, S, incorporates both exclusive-or and addition operations subsequent to the s-box lookup. The enforcement of this requirement avoids the weaknesses described above.

4. (a) The xor difference of each distinct pair of entries in S is unique; and
(b) The subtraction difference of each distinct pair of entries in S is unique.

The design of a secure s-box seeks to minimize the frequency of output differences which increases the likelihood of a lower differential probability. Specifically, requirement 4.(a) attempts to achieve an s-box difference distribution table with low differential uniformity, which contributes towards reducing the existence of high probability differential characteristics. A vulnerability, that may be exploited when s-boxes possess repeated subtraction output differences, is avoided by satisfying requirement 4.(b).

5. Each distinct pair of entries in S differs in at least four bits.

This condition is the weakest of the 5 differential requirements and appears to serve simply to prevent excessive numbers of corresponding bits in distinct pairs of s-box entries. In particular, it aids in preventing the careful placing of a number of zero bits in positions of an output difference which may assist cryptanalysis attempts.
Chapter 6. Practical Application of Heuristic Techniques to MARS-Like S-Box Generation

When outlining the MARS s-box linear property requirements below, we utilize the correlation matrix of an s-box [20] as a useful means of relating the linear requirements imposed by the designers of the MARS s-box with those columns of its correlation matrix which are being positively affected by the existence of these specific requirements.

Linear Property Requirements

1. Parity Bias: The parity bias of $S$, $|Pr_x[\text{parity}(S[x]) = 0] - \frac{1}{2}|$ is required to be at most $\frac{1}{32} = 0.03125$.

The parity bias requirement places a limit on the inequality between the number of zero and one parity bits totalled over all entries in the s-box. This MARS s-box requirement permits a deviation of a 16 bit bias at most in either direction from a probability of $\frac{1}{2}$ which equates to a valid range of between 240 and 272 zeros or ones inclusive. In terms of the linear correlation matrix, the only matrix column that is relevant to this requirement is the column which corresponds to $S[x_0] \oplus S[x_1] \oplus ... \oplus S[x_{31}]$, the xor sum of all the output bits for each s-box entry. The absence of this requirement may lead to an excessive parity bias. In this case, the expected workload for a linear cryptanalytic attack would be significantly reduced as the parity bit is a function of all bits in an s-box entry.

2. Single-bit Bias: The single-bit bias of $S$, $|Pr_x[S[x]_i = 0] - \frac{1}{2}| \forall i \in \{0,..,31\}$, is required to be at most $\frac{1}{30} \approx 0.03333$.

This requirement places a restriction on the imbalance of each of the individual boolean functions which comprise the output entry. In a situation where no single-bit bias exists, we have $\#(S[x]_i = 0) = \#(S[x]_i = 1) \forall x \in \{0,..,511\}, \forall i \in \{0,..,31\}$. The MARS designers have allowed a maximum deviation of $\frac{1}{30}$ either side of this. The reason for this particular maximum bias value is unclear as it does not represent a whole number of bits out of 512 ($2^9$). The single-bit bias above can be calculated by computing the hamming weight of each of the individual output functions of $S$. Therefore, the single-bit bias requirement considers the 32 single term columns of the linear correlation matrix. As some cryptanalytic attacks may trivially involve approximations of component functions based on constant functions, minimizing the single-bit bias reduces the likelihood of this occurring.
3. **Two Consecutive Bits Bias:** The two consecutive bits bias of S,
\[ |Pr_x[S[x]_i \oplus S[x]_{i+1} = 0] - \frac{1}{2}| \forall i, 0 \leq i \leq 30, \]
is required to be at most \( \frac{1}{30} \approx 0.03333 \).

This requirement limits the hamming distance between adjacent output boolean functions away from \( 2^{9-1} \). In the case of the MARS s-box, S, the allowable margin is \( 2^{9} \frac{1}{30} \) either side of a hamming distance of \( 2^8 \). Again, note that \( \frac{1}{30} \) does not represent a whole number of bits and the justification for this choice of value is not expressed in the MARS documentation. We conjecture that this choice was strongly encouraged by the bias achieved by the generated s-box being close to this value. As there are 31 adjacent pairs of boolean functions in the output, the two consecutive bits bias affects 31 columns of the linear correlation matrix. In terms of linear cryptanalysis, limiting the two consecutive bits bias makes it difficult to find a subset of adjacent output bit pairs with high probability bias. This, in turn, decreases the likelihood of successfully constructing a good linear approximation of the s-box based on such a subset.

4. **Single-bit Correlation:** The single-bit correlation of S,
\[ |Pr_x[S[x]_i = x_j] - \frac{1}{2}| \forall i, j (0 \leq i \leq 31, 1 \leq j \leq 9) \]
is to be minimized.

This requirement seeks to minimize the single-bit correlation of the s-box. Unlike the previous conditions, the MARS designers did not set a bound on this requirement. The single-bit correlation affects 32 columns of the linear correlation matrix as each bit of the s-box entries are correlated independently of each other with the input bits. This property minimizes the probability bias for linear approximations of the s-box which comprise a single input bit and a single output bit.

Table 6.1 shows the extent to which the above 9 differential and linear requirements were satisfied by the MARS s-box at the time this work was performed. Note that D1 to D5 indicates differential requirements 1 to 5, and L1 to L4 indicates linear requirements 1 to 4, as described above.

As can be seen from Table 6.1, differential requirements 1, 3 and 5 are satisfied by the 9x32 s-box, S. Differential requirement 2 is satisfied by the 8x32 s-boxes, S0 and S1. An analysis of the differential properties of S revealed that it does not, however, satisfy differential requirement 4. This is because a number of equal xor and subtraction differences exist within S. Rather than the expected
Table 6.1: MARS s-box: Satisfaction of Differential and Linear Property Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Bound</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D2</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D3</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D4</td>
<td>-</td>
<td>Requirement Not Satisfied</td>
</tr>
<tr>
<td>D5</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>L1</td>
<td>$\frac{1}{32} = 0.03125$</td>
<td>0.0078125</td>
</tr>
<tr>
<td>L2</td>
<td>$\frac{1}{30} \approx 0.03333$</td>
<td>0.033203</td>
</tr>
<tr>
<td>L3</td>
<td>$\frac{1}{30} \approx 0.03333$</td>
<td>0.03125</td>
</tr>
<tr>
<td>L4</td>
<td>-</td>
<td>0.044922</td>
</tr>
</tbody>
</table>

130816 distinct xor differences and 2 x 130816 distinct subtraction differences, it can be shown that S has 130813 distinct xor differences and 2 x 130808 distinct subtraction differences. The repeated differences are set out below. In each equation, the xor/subtraction difference of the indexed words on the left is equal to the xor/subtraction difference of the indexed words on the right.

\[
S[27] \oplus S[292] = S[101] \oplus S[360] \\
S[27] \oplus S[101] = S[292] \oplus S[360] \\
S[27] \oplus S[360] = S[101] \oplus S[292] \\
\]

With regard to the linear requirements, the parity bias of S is significantly lower than the set bound of $\frac{1}{32}$. The single-bit bias and two consecutive bits bias are both marginally below the limit of $\frac{1}{30} \approx 0.03333$. The maximum single-bit correlation bias of S is about 0.044922 < 0.0454545, as stated in the MARS paper. Thus all linear conditions imposed by the designers of the MARS s-box are satisfied by S.
6.1.4 Technique used for MARS S-Box Generation

The designers of the MARS s-box used the well known SHA-1 (Secure Hash Algorithm-1) [75] to generate their 9x32 s-box, S. The output of SHA-1 is a 160-bit digest comprised of the concatenation of five 32-bit words. The input used for SHA-1 to obtain entries of the s-box is the value \( 5i|c1|c2|c3 \) where \( i = 0,..,102 \) and \( c1 \) and \( c2 \) are the fixed constants

\[
\begin{align*}
c1 &= 0xb7e15162 \\
c2 &= 0x243f6a88
\end{align*}
\]

The parameter \( c3 \) is a sequentially incremented seed that varies until all five differential properties and linear properties 1 - 3 are satisfied. Satisfaction of the first eight property requirements, together with a satisfactory minimization of linear property 4, determined the final value of \( c3 \). Thus, entries of the 9x32 s-box, S, were computed as follows:

\[ S[5i+k] = SHA-1(5i|c1|c2|c3) \quad (k = 0,..,4, i = 0,..,102) \]

where \( k \) denotes the \( k^{th} \) output word of SHA-1.

The designers of the MARS s-box began the computational process of generating S, using SHA-1, with \( c3 = 0 \), increasing \( c3 \) until the final s-box was found. Each value of \( c3 \) resulted in a 9x32 s-box which was divided into two 8x32 s-boxes. For each value of \( c3 \), the xor sum of distinct pairs in S0 and S1 was checked to see if it contained more than one zero byte. If this was the case, then \( S[i] \) was replaced by \( 3 \cdot S[i] \) for one of the words \( S[i] \) in the pair. The new s-box was again tested for the five differential requirements and linear requirements 1 - 3. If this test was passed then the single-bit correlation was calculated. The final fixed constant value of \( c3 \) was 0x02917d59. This value was found to best minimize the single-bit correlation. As stated in [39], the program for generating S ran for about a week, with the value of \( c3 \) increasing to \( 0x02917d59 = 43\,086\,937_{10} < 2^{26} \). The MARS s-box can be found in [39].
6.2 MARS-like S-Box Generation

The idea to conduct research work into an alternative way of producing MARS-like s-boxes arose largely from the seemingly excessive computational effort of producing the MARS s-box, particularly given the properties it exhibited. A consequence of experience with earlier s-box generation experiments, as described in Chapter 5 of this thesis, was a good awareness of the typical run times of generating s-boxes of various sizes with particular cryptographic properties using existing heuristic techniques. Thus, with reasonable certainty, experience told us that s-boxes of this size possessing some good properties could be generated much quicker. However, it was uncertain whether s-boxes heuristically generated in this way would satisfy all the differential and linear requirements placed on the MARS s-box. The focus of the research centred on analysis and experimentation with cryptographic properties of s-boxes and their effect with respect to the imposed differential and linear requirements. Although not expected to be an issue, an ancillary consideration to the research was the efficiency of the chosen heuristic method for generating the s-boxes.

6.2.1 MARS-like S-Box Property Requirements

In an attempt to achieve satisfaction of the differential and linear conditions imposed on the MARS s-box, a small set of different property requirements were specified in the generation process of the MARS-like s-boxes. This component of the research stemmed from a hypothesis that the generation of generically strong s-boxes would produce s-boxes that were either in compliance with the MARS requirements, or were capable of easy adaptation to meet the requirements. This hypothesis was initially supported by the knowledge that it was possible to generate cryptographically strong s-boxes using heuristic techniques in a comparatively short amount of time. In particular, the conditions which we placed on the MARS-like s-boxes were chosen for their known contribution to resisting linear cryptanalysis. Their use was also experimental in that, firstly, we were initially unsure whether they were sufficient to fully satisfy the MARS linear requirements and, secondly, the ability to subsequently meet the differential requirements of the MARS s-box without violating the achieved linear requirements was an interesting research challenge.

Our first requirement for the MARS-like s-boxes was that all 32 boolean func-
tions comprising an s-box were balanced. Any deviation from balance can be considered both numerically and conceptually identical to the existence of a single-bit bias, which is the subject of MARS linear requirement 2. Thus, a balanced boolean function equates to a single-bit bias of zero.

An important restriction which was included in the computational process was the selection of a minimum allowable nonlinearity value for each boolean function comprising the MARS-like s-box. Note that this condition does not directly map to any of the MARS linear requirements. Nonlinearity and correlation immunity are known to be conflicting properties and, as such, if low order correlation immunity exists then a high nonlinearity is possible. The optimization of the nonlinearity property affects the 32 single term columns of the linear correlation matrix in terms of the magnitude of the highest value in each column. This property was considered most significant in the MARS-like process as it directly influences the success of linear cryptanalytic attacks, as discussed in Section 2.3.2.

The third parameter chosen to generate MARS-like s-boxes was a limit on the maximum imbalance between distinct pairs of boolean functions comprising the s-box. In other words, the hamming distance between each distinct pair of boolean functions was bounded and thus not permitted to deviate too much from $2^{9-1}$. This condition encompasses the MARS linear requirement 3, two consecutive bits bias, in that it includes adjacent pairs of boolean functions in the s-box. Therefore, by imposing this restriction on MARS-like s-boxes, this represents a slightly greater contribution to its overall resistance to linear cryptanalysis.

The generation process of MARS-like s-boxes relied on a fourth requirement to reduce the correlation between input bits and output bits of the s-box. This parameter specified a maximum deviation from CI(1) for all 32 boolean functions in the s-box. This can be viewed as a condition which is identical to the MARS linear requirement 4, single-bit correlation, with the additional constraint that a bound on the allowable deviation from CI(1) has been set, rather than a desire to minimize.

We have set out above the four property requirements imposed in order to generate MARS-like s-boxes. As stated above, these focussed largely on achieving satisfaction of the MARS linear properties thus implicitly strengthening their resistance to linear cryptanalytic attacks. The extent to which the MARS differential and linear property requirements were satisfied by our generation process is discussed in Section 6.2.3.
6.2.2 Technique used for MARS-like S-Box Generation

Before presenting the experimental results, we first discuss the specific heuristic technique and approach that was used in this research for the generation of MARS-like s-boxes. The core of this generation process was the use of the Hill Climbing Method which is described in Section 3.1.1 of this thesis. For ease of reference, we outline below the key steps of the Hill Climbing Method for generating boolean functions:

1. Measure the property to be optimized for the original function.
2. Select a pair of elements $i, j$ to complement ensuring that $i \neq j$.
3. Perform the swap to produce a new function.
4. Measure the relevant property for the new function.
5. If the property measure has improved, replace the original function with the new function. If not, retain the original function.
6. Repeat steps 2 - 5 until a predetermined stopping criteria has been reached.

The primary goal of the Hill Climbing Method in the MARS-like s-box generation process was to generate component boolean functions with at least the minimum required nonlinearity only. Consequently, none of the other MARS-like s-box property requirements were directly targeted by the hill climbing process (although balance was maintained) but were subsequently considered. The choice of hill climbing as the heuristic method utilized in the MARS-like s-box generation was mainly due to its computational efficiency in comparison with other heuristic methods capable of achieving similar property values, at the time this research was performed.

We now describe the general procedure for constructing 9x32 MARS-like s-boxes. Recall that certain property requirements were placed on the two individual 8x32 s-boxes which comprise the MARS s-box, $S$. For this reason, the approach we took was to firstly generate 8x32 s-boxes which satisfied the four MARS-like conditions stipulated above. Pairs of s-boxes of this size were combined to form a 9x32 s-box.

The procedure began with the random generation of single output balanced boolean functions. Each boolean function was hill climbed in order to achieve the
minimum nonlinearity value specified in the code. Candidate 8-variable functions possessing at least this nonlinearity value were kept for further processing. From these functions, only those satisfying the specified maximum deviation limit from Cl(1) were retained. Subsequently, a process of cumulative construction based on retention of functions undergoing a progressive pairwise analysis to exclude those exceeding the maximum imbalance limit was performed. A set of 32 8-variable boolean functions achieving these limits comprised an 8x32 s-box containing 256 words. A 9x32 s-box was formed by combining two 8x32 s-boxes generated by this process.

The constructed 9x32 MARS-like s-box was then checked for the differential and linear requirements placed on the MARS s-box. In particular, differential requirement 1 was firstly checked. In the event of either or both the all zeros or all ones word existing as entries in the s-box, then the procedure was to replace such entries with random words. In order to satisfy differential condition 2, if more than one byte within any number of words, \( w \), in an 8x32 s-box was equal and in the same position, a process of byte replacement with random bytes would take place in \( (w - 1) \) number of entries in the s-box. For differential requirement 3, a check for violation of the three conditions in the 9x32 s-box was performed. A failure to satisfy any aspect of this requirement was to be dealt with by replacing the violating entries with random words in the 9x32 s-box. Upon ascertaining whether differential requirement 4 was achieved by the 9x32 s-box, the action to be taken to establish the uniqueness of the xor and subtraction differences involved replacement of the second entry which caused the same difference in the s-box with a random entry. A determination of whether differential requirement 5 was met by the 9x32 MARS-like s-box was made. Note that within each requirement, before accepting a random replacement of bytes or words, the candidate random byte or word was checked for compliance with that particular requirement only. This process was repeated, if necessary. Subsequent to acceptance of any modified entries, the new s-box was tested for all nine conditions again. We ensured that the introduction of any replacement entries in the s-box did not destroy the balance property achieved by the initial functions. The process used in dealing with each of the four MARS linear conditions was to simply check for compliance with each requirement and, if one or more bounds had been violated, discard the s-box.
A summary of the above steps of the MARS-like generation procedure is presented in Algorithm 6.2. Note that $NL_{\text{min}}$ is the minimum acceptable nonlinearity value required to be exhibited by each component boolean function. The parameter, $\text{maxCIdev}$, represents the maximum allowable deviation from first order correlation immunity. $\text{maxhddev}$ is the maximum allowable deviation from balance between distinct pairs of stored functions.

Algorithm 6.2: Generation Process for MARS-Like S-Boxes

1. Generate multiple 8x32 s-boxes, each in the following manner:
   
   (a) Let $k = 0$.
   (b) Generate a random 8-variable balanced boolean function, $f(x)$.
   (c) Perform strong hill climbing on $f(x)$ to produce $g(x)$.
   (d) If $NL(g) < NL_{\text{min}}$, reject $g(x)$ and return to Step (b).
   (e) If $\hat{G}(\omega) > \text{maxCIdev}$ for any $\omega$ where $hw(\omega) = 1$, reject $g(x)$ and return to Step (b).
   (f) If $k > 1$ and $hd(g, h) > \text{maxhddev}$ for any previously accepted function, $h(x)$, reject $g(x)$ and return to Step (b).
   (g) Accept and store $h(x) = g(x)$ and increment $k$.
   (h) If $k < 32$, return to Step (b).

2. Combine a pair of 8x32 s-boxes to form a 9x32 s-box, $SB[i]$, $0 \leq i \leq 511$.

3. Check $SB$ for compliance with MARS differential requirements. Replace any violating portion of or whole $SB[i]$ with a random byte or word appropriately which is not in violation of its specific MARS differential requirement nor in violation of the MARS-like balance requirement.

4. Re-check $SB$ for compliance with MARS differential requirements.

5. Assess modified s-box against MARS linear requirements. If any linear requirement is not met, discard this pair of s-boxes and commence Step 2 with another pair of 8x32 s-boxes.

Alternative approaches to the generation of MARS-like s-boxes may be taken by varying the technique which we used. A possible improvement to the process could be made by application of a different heuristic technique which is also suit-
able for optimizing one or more of the properties known to enhance resistance to linear and/or differential cryptanalysis. Heuristic techniques such as Genetic Algorithms [65] have been demonstrated to be successful in generating cryptographically strong boolean functions and s-boxes with these types of properties. Also, Method 1 and Method 3, proposed in this thesis, are further examples of alternative heuristics which could be used to generate the component boolean functions of MARS-like s-boxes. A further variation to our approach would be to include a number of extra parametric constraints in the generation process to achieve a stronger s-box with additional desired target properties, for example, some avalanche criteria. MARS-like s-boxes exhibiting a different emphasis on their existing cryptographic criteria can be generated by appropriately varying the parameters. Changing the emphasis can be useful when it is desired to create MARS-like s-boxes designed for optimality in certain properties and remaining properties being suboptimal will not compromise the security of the s-box. This is, of course, restricted by the principles involving co-existence of conflicting properties.

The MARS s-box linear requirements affect 64 columns of its linear correlation matrix. The requirements which we impose in our MARS-like s-box generation process affect 528 columns of the linear correlation matrix. As a slightly increased number of columns of the linear correlation matrix are influenced by our choice of requirements for MARS-like s-box generation, the MARS-like s-boxes tend to be more resistant to linear cryptanalysis in particular. There are a total of $2^{32}$ columns in the linear correlation matrices of these s-boxes. To calculate and then optimize a complete linear correlation matrix of this size is not practical due to the computational effort required for this task. However, an adaptation to our MARS-like generation process which considers a greater number of columns of the matrix will result in the generation of even stronger s-boxes.

### 6.2.3 Experimental Results

In this section we discuss whether the four conditions and general technique of generating MARS-like s-boxes was successful in meeting the nine criteria defining a MARS s-box, thus possessing at least the same properties and level of strength against differential and linear cryptanalysis. Further, we compare the characteristics of these two types of s-boxes, each created by their specific requirements. We also compare the performance of the processes involved in creating MARS
Chapter 6. Practical Application of Heuristic Techniques to MARS-Like S-Box Generation

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Bound</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D2</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D3</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D4</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D5</td>
<td>-</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>L1</td>
<td>$\frac{1}{32}$</td>
<td>0.03125</td>
</tr>
<tr>
<td>L2</td>
<td>$\frac{1}{32}$</td>
<td>$\approx$ 0.03333</td>
</tr>
<tr>
<td>L3</td>
<td>$\frac{1}{30}$</td>
<td>$\approx$ 0.03333</td>
</tr>
<tr>
<td>L4</td>
<td>-</td>
<td>0.03125</td>
</tr>
</tbody>
</table>

Table 6.2: MARS-like s-box: Satisfaction of MARS Differential and Linear Property Requirements

and MARS-like s-boxes.

We outlined in Section 6.1.3 the differential and linear property requirements imposed by the designers of the MARS s-box. We now discuss the extent to which these conditions were able to be satisfied by an example MARS-like s-box. We refer to our example 9x32 MARS-like s-box as SB and the two 8x32 s-boxes of which it is comprised as Sb1 and Sb2 respectively. Note that SB[i] (i = 0,...,511) contains 512 32-bit words and Sb1[j] and Sb2[j] (j = 0,...,255) each contain 256 32-bit words. SB can be found in Appendix B.

SB automatically satisfied differential conditions 1, 3 and 5, without the need for any word replacement. In order to satisfy differential condition 2, it was necessary to modify a small number of bytes in each of the 8x32 s-boxes, Sb1 and Sb2, typically in less than half a dozen entries, and re-checking that condition, particularly for previous pairs of entries. Similarly, the satisfaction of differential requirement 4 involved replacing a small number of entries in the 9x32 s-box, SB. Thus, four of the differential requirements placed on the MARS s-box were satisfied by our example MARS-like s-box, SB. In addition, Sb1 and Sb2 both satisfied the condition placed on the 8x32 s-boxes individually, namely, differential requirement 2. This situation is reflected in Table 6.2.

Table 6.2 shows the extent to which the MARS differential and linear requirements were satisfied by our example MARS-like s-box. Note that D1 to D5 indicates MARS differential requirements 1 to 5, and L1 to L4 indicates MARS linear requirements 1 to 4, as described in Section 6.1.3.

We make the following observations from Table 6.2. The parity bias of SB is...
6.2. MARS-like S-Box Generation

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Bound</th>
<th>Achieved: MARS</th>
<th>Achieved: MARS-like</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>-</td>
<td>Requirement Satisfied</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D2</td>
<td>-</td>
<td>Requirement Satisfied</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D3</td>
<td>-</td>
<td>Requirement Satisfied</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D4</td>
<td>-</td>
<td>Requirement Not Satisfied</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>D5</td>
<td>-</td>
<td>Requirement Satisfied</td>
<td>Requirement Satisfied</td>
</tr>
<tr>
<td>L1</td>
<td>(\frac{1}{32}) = 0.03125</td>
<td>0.0078125</td>
<td>0.019531</td>
</tr>
<tr>
<td>L2</td>
<td>(\frac{1}{30}) \approx 0.03333</td>
<td>0.033203</td>
<td>0.0</td>
</tr>
<tr>
<td>L3</td>
<td>(\frac{1}{30}) \approx 0.03333</td>
<td>0.03125</td>
<td>0.023438</td>
</tr>
<tr>
<td>L4</td>
<td>-</td>
<td>0.044922</td>
<td>0.03125</td>
</tr>
</tbody>
</table>

Table 6.3: MARS and MARS-like s-boxes: Satisfaction of Differential and Linear Property Requirements

0.019531 which is less than the threshold value of \(\frac{1}{32}\) = 0.03125. The single-bit bias of SB is zero, which signifies that each of the 32 boolean functions, which comprise the 9x32 s-box, are balanced. The two consecutive bits bias of SB is at most \(0.023438\) which is less than the bound of \(0.033333\). The maximum single-bit correlation bias of SB is 0.03125 < 0.0454545. Consequently, all linear conditions imposed by the designers of the MARS s-box are satisfied by our example 9x32 s-box, SB.

The results presented in Tables 6.1 and 6.2 are now amalgamated in Table 6.3 for the purposes of comparison of results. As previously stated, D1 to D5 indicates MARS differential requirements 1 to 5, and L1 to L4 indicates MARS linear requirements 1 to 4, as described earlier.

Table 6.3 shows that all of the MARS differential requirements were able to be satisfied by our MARS-like s-box. The MARS s-box itself, however, was shown to not satisfy differential requirement 4 in that a number of xor and subtraction differences in the s-box were found to be repeated and thus did not achieve the required absolute uniqueness. The MARS s-box was able to satisfy all other differential property requirements.

From Table 6.3, we observe that all linear requirements were met by both our MARS-like s-box and the MARS s-box. When comparing linear requirements 2, 3 and 4 it is clear that the magnitude of the bias of our MARS-like s-box is noticeably smaller than that possessed by the MARS s-box, S. This is due to three of the four MARS-like conditions influencing these particular linear properties. The achievement of all MARS-like linear results depended largely on the four
parameters used in our s-box generation program.

Table 6.3 shows that the MARS-like s-box, SB, has a single-bit bias of zero. An s-box comprised of balanced boolean functions clearly possesses no single-bit bias since there are an equal number of ones and zeros in the truth tables of balanced boolean functions. Thus, there is no output bias in the individual component boolean functions to be exploited. Our s-box generation procedure began with the generation of a set of 32 balanced boolean functions. In order to satisfy differential requirements 2 and 4 it was necessary to replace a small number of bytes and entries respectively. However, at all times throughout our computations we retained balance in the boolean functions. We now contrast this with the MARS s-box, S. None of the boolean functions comprising the MARS s-box are balanced. The extent of the single-bit bias over the 32 output functions ranged from 1 bit to 17 bits out of $2^9 = 512$. This deviation from balance just falls within their given bound. Again, a bound of $\frac{1}{30}$ does not reflect a deviation from a balanced function of a whole number of bits.

Nonlinearity is a very important cryptographic property of single output boolean functions and s-boxes. High nonlinearity provides increased resistance to linear cryptanalysis by reducing the magnitude of the entries in the linear approximation table which, in turn, increases the difficulty of finding a linear approximation with high probability. The nonlinearity of an s-box is a function of the magnitude of the largest value in the linear correlation matrix. The measure of s-box nonlinearity is the maximum nonlinearity value taken over all linear combinations of the output boolean functions which comprise the s-box. As there are 32 output boolean functions in the MARS s-box, resulting in $2^{32} - 1$ linear combinations, the task of optimizing the nonlinearity of an s-box of this size is computationally infeasible. However, for the MARS-like s-box we achieved a partial enforcement of the nonlinearity goals through specifying a parameter that ensured a minimum nonlinearity for each of the 32 boolean functions comprising the s-box. In contrast, no requirements on nonlinearity values were given by the MARS s-box designers. For the 8x32 MARS-like s-boxes, the nonlinearity of the individual boolean functions in Sb1 and Sb2 ranged from 108 to 112 inclusive, with an average nonlinearity of 110. The boolean functions comprising the 8x32 MARS s-boxes, S0 and S1, have nonlinearity values ranging from 92 to 109, the most frequent nonlinearity value being 102. For our experiments, we typically generated sets of 32 boolean functions with a minimum nonlinearity of
110, although we experimented with parameters above and below this value. A parameter value for minimum nonlinearity at around 108 produced 8x32 s-boxes in less than 10 minutes, while minimum nonlinearities of 112 for an 8x32 s-box took about 3 to 4 hours to generate.

Although in our MARS-like s-box generation procedure we have not directly sought to optimize autocorrelation, the boolean functions comprising our s-boxes have, in general, displayed low maximum absolute autocorrelation values. A low autocorrelation distribution for an s-box (even for only its component boolean functions) serves to improve its differential properties, in particular, by helping to reduce its differential uniformity. The range of maximum absolute autocorrelation values for the individual boolean functions in our 8x32 s-boxes, Sb1 and Sb2, was between 48 and 88, averaging around 56. We note that the boolean functions comprising the MARS s-boxes, S0 and S1, displayed maximum absolute autocorrelation values of between 52 and 88, averaging 64.

One of the more important parameters set by our code was a limit on the maximum imbalance of the xor sum of distinct pairs of boolean functions. The purpose of this restriction on boolean function candidates was to reduce the imbalance between all distinct pairs of boolean functions in the s-box. This restriction not only results in a reduction in the two consecutive bits bias as imposed by the MARS s-box requirements, but additionally reduces the imbalance between all other pairs of distinct boolean functions in the s-box, whether adjacent or otherwise. Thus, our condition represents a more rigorous strengthening of the s-box in terms of pairwise bit bias. Typical values for this parameter in the generation process used to limit the maximum imbalance between distinct pairs of boolean functions in the set were 10 and 12.

In order to manage the single-bit correlation requirement placed on the MARS s-box, we included a requirement in our generation process that achieved an identical outcome. By setting a maximum deviation from CI(1) for each of the 32 individual boolean functions in the MARS-like s-box, we were easily able to satisfy MARS linear requirement 4, single-bit correlation. Minimizing this deviation reduces the magnitude of correlations which exist between individual input and individual output bits of the s-box. Typically, we used parameter values of either 16 or 24 to be the maximum allowable deviation from CI(1) for the 32 boolean functions comprising the 8x32 s-boxes.

A combination of hill climbing and appropriate setting of the parameters dis-
cussed above allowed us to produce good 8x32 s-boxes, pairs of which gave us 9x32 MARS-like s-boxes, not only satisfying all of the MARS differential and linear requirements, but also possessing better nonlinearity and autocorrelation values. Most of the s-boxes generated by our technique were very close to satisfying the MARS s-box requirements. In fact, for those s-boxes which we successfully generated, bias values, when exceeded, were so by only extremely small margins. The remaining MARS-like s-boxes which we generated were easily able to satisfy the same conditions that the MARS s-box satisfied.

Using a heuristic technique approach, our MARS-like s-box generation process was based on the Hill Climbing Method, due to the efficiency of generation previously established in our earlier experiments. This approach, together with some carefully chosen parameters, enabled us to generate a number of MARS-like s-boxes with little effort. The program execution time varied from approximately 16 minutes to around 3 hours and 20 minutes on a single Pentium II 300 MHz PC for 8x32 s-boxes. This range in the execution time was dependant on the stringency of the parameters chosen. The task of concatenating pairs of 8x32 s-boxes to form 9x32 MARS-like s-boxes was relatively instantaneous in terms of time. Our time frame for the entire MARS-like s-box generation process represented a highly significant improvement on the program running time for the MARS s-box of about a week.

6.3 Summary

In this chapter we have outlined an alternative approach to the generation of MARS-like s-boxes providing satisfaction of all of the requirements placed on the MARS s-box. Our research has clearly demonstrated that by using a combination of random boolean functions, heuristic techniques and appropriate parameters, we not only easily satisfied all the MARS differential conditions and fell inside the specified bounds of all the MARS linear conditions, but also gained additional properties in the component boolean functions such as higher nonlinearity and balance.

Generating the MARS s-box involved a long search running through values for $c_3$ and taking the MARS designers about a week to produce the single final s-box. Not only did our technique produce a cryptographically superior s-box but did so in a fraction of the time, that being at most a few hours. Furthermore,
our generation process is capable of producing not only one, but a number of MARS-like s-boxes.
Chapter 7

Conclusions

To conclude this thesis, a summary of the research performed for this thesis is presented in the first section of this chapter. This is followed by a discussion of possible future directions that could extend this research in the second and final section of this chapter.

7.1 Summary of Thesis

The primary goal of the research work reported in this thesis was to apply heuristic techniques for the optimization of boolean function and s-box properties, with the consequent aim of improving the security of ciphers which incorporate such components. The necessary background theory pertaining to boolean functions and s-boxes, and relevant prior research work performed by other researchers, have been outlined in preceding chapters. A description of the existing heuristic techniques applied in this thesis was provided.

In this thesis, we have proposed three new heuristic methods to be used for improving the cryptographic properties exhibited by boolean functions. Each of the three methods have been developed to focus on a different significant strengthening property, but is designed to optimize a combination of properties. These methods are novel, flexible and elegant, and were all successful in achieving their respective intended outcomes effectively and reasonably efficiently.

The first method recursively applied dynamically selected small, incremental changes to a bent boolean function. The hypothesis proposed for this new work,
that highly nonlinear balanced functions with low autocorrelation can be generated by searching the boolean function space in close hamming proximity to bent functions, was proven correct. This method was extremely successful in efficiently generating high quality balanced boolean functions exhibiting high nonlinearity and low autocorrelation, including some examples of the best known functions at that point in time, some of which still represent the best known combination of property values discovered. Further, just as importantly, none of the generated functions possessed non-zero linear structures.

The basis for the second new heuristic method presented in this thesis is the careful controlling of the magnitude and zero positions of values in the Walsh Hadamard transform vector of pairs of lower dimensional boolean functions to be concatenated to form higher dimensional functions. The successive building of target higher dimensional resilient boolean functions with high nonlinearity using experimental parameters was shown to be successful in generating multiple such functions with optimal algebraic degree for a range of orders of resilience. Of particular note in this second new method was the ease by which the conflicting cryptographic properties of nonlinearity and resilience could be simultaneously optimized. Many examples of boolean functions with almost all of the then best known optimal values for these combined properties were able to be generated by this method.

A third new heuristic method developed for this thesis involved intelligent bit manipulation of the binary starting function in order to generate balanced boolean functions exhibiting some non-zero degree of propagation criteria, as well as high nonlinearity. The starting functions for the computational process needed to satisfy certain criteria. To this end, two new generation schemes were developed to produce the starting functions for this method. Further, these starting function generation schemes may be utilized in other heuristic methods, or possibly construction methods, to obtain suitable initial functions. An optional feature of this third new method was the ability to optimize the algebraic degree of the final function. This method was successful in generating examples of balanced $PC(k)$ boolean functions with good nonlinearity possessing some of the best combinations of these complementary properties that have been reported by other methods. Again, experimental results produced no functions containing non-zero linear structures. Additionally, whilst not a specific target of the method, functions which could, at the same time, be linearly transformed into
first-order correlation immune functions were discovered with relative ease.

The existing heuristic techniques of Genetic Algorithm, Hill Climbing Method and combined Genetic Algorithm/Hill Climbing were applied in this research to randomly generate $N \times M$ regular s-boxes $N > M$, with experiments conducted for s-boxes with $N = 8$ input bits with $M \in \{2,3,4,5,6,7\}$. The goal of experiments was the improvement in the nonlinearity and autocorrelation values for s-boxes of these sizes, with experiments concentrated on 8x4 regular s-boxes. It was clearly demonstrated that these existing heuristic techniques were able to produce $N \times M$ regular s-boxes with not only significantly improved nonlinearity and autocorrelation values but also a greater quantity of such s-boxes than compared with randomly generated $N \times M$ regular s-boxes. This research work has progressed the ongoing investigation into ways in which stronger s-boxes may be obtained for possible incorporation into cryptographic ciphers.

The Hill Climbing Method was applied to generate component boolean functions, a selection of which were subsequently used in a structured generation process, in order to construct 8x16 and 8x32 s-boxes which met specific property criteria. In this new work, further constraints were placed on the relationship between component functions of the s-boxes in addition to the nonlinearity optimization requirement. The successful outcome of this work resulted in the incorporation of two of the generated s-boxes into a number of the stream ciphers from the SOBER family of stream ciphers [80], [36], [81]. Thus, the practicalities of using a heuristic technique such as the Hill Climbing Method, combined with a defined generation process and suitable parameters ascertained by experimentation, have been demonstrated to be successful by the generation of numerous s-boxes suitable for use as cipher components.

The earlier s-box work undertaken for this thesis provided a good understanding of the tradeoffs which exist between achievable properties and computational effort of the three existing heuristic techniques applied in the above work. This, in large part, led to the motivation for investigating the properties of the 9x32 s-box used in the MARS block cipher [39]. These MARS s-box properties were analyzed against the requirements stated in the MARS design documents. Experiments using the Hill Climbing Method in the generation of similar 9x32 s-boxes satisfying the same MARS s-box requirements were performed with outstanding success. It was found that this MARS-like s-box generation approach was not only able to generate 9x32 s-boxes with better cryptographic properties, but was
also significantly more computationally efficient by several orders of magnitude. This new research work further demonstrated the practical usefulness and effectiveness of heuristic methods in obtaining strong components for incorporation into functional ciphers.

7.2 Future Directions

During and subsequent to the research performed for this thesis, a number of areas of future work have been identified. We now discuss directions for future research which involve both an extension of some of the work contained in this thesis, as well as topics of related work which could be investigated.

The development of new heuristic methods for application to both boolean functions and s-boxes to improve their cryptographic properties is an important area of necessary research. This will continue to be so due to the ever increasing need to incorporate strong components into cipher systems to enhance their resistance to cryptanalytic attacks. More specifically, the development of future methods should further aim to simultaneously optimize multiple cryptographic properties which will be dependent on the limitations of property combination co-existence. Of equal importance should be the development of methods which have the ability to efficiently perform directed searches through larger search spaces, and thus producing strong functions with a higher number of input variables. Particularly for large inputs, consideration should also be made to the satisfaction of some cryptographic properties to a “close enough” degree which, although secondary to being able to achieve full satisfaction of such properties generally attainable for lower numbers of inputs, will still provide some level of strength against cryptanalytic attacks.

Heuristic methods have been demonstrated in this thesis to be of significant practical use for the cryptographic property optimization of single and multiple output boolean functions. Possible extensions for future research exist for each of the new heuristic methods developed for this thesis. These have been identified in the appropriate sections of Chapter 4. Additionally, an obvious extension to the research work conducted in Chapter 4 of this thesis would be the application of each of the three new heuristic methods to $N \times M$ s-boxes to enhance the cryptographic properties targeted by each of the respective methods.

New methods for constructing strong boolean functions and s-boxes which
are a combination of heuristic techniques and algebraic constructions would be a worthwhile direction of research. The rationale for this is that inherent limitations of each technique may possibly be overcome by combining the two approaches together to work towards capitalizing on their advantages. Heuristic techniques tend to be able to produce a large number of functions with good cryptographic properties but become inefficient for very large numbers of inputs. Algebraic constructions, on the other hand, typically enable the construction of a small number of functions with optimal or close to optimal properties, often with potential weaknesses in structure, while the number of inputs tends to be of less relevance than with heuristic searches. Two possible options can be considered if these approaches are to be combined. Firstly, heuristic techniques may be applied to generate a large number of good functions, a selection of which could be used to algebraically construct multiple functions with optimal properties and possibly without the structural weaknesses. As a second option, algebraic constructions may be used to construct initial functions for commencing a heuristic technique to enable the search to focus on very specific areas of the space which exhibit desired properties.

The work done in Chapter 6 of this thesis has demonstrated that the independent analysis of properties of S-boxes incorporated into existing cipher systems is a useful task in that the strengths and weaknesses of such components may be ascertained and, if necessary, investigations into their improvement using heuristic methods can represent valuable research in the field. Thus, the analysis and investigation of security attributes of components of existing currently used cipher systems would be a worthwhile topic of future research.
Appendix A

Example 8x32 S-Box and Analytical Information

Example 8x32 S-box:

0x1faa1887, 0x4e5e435c, 0x9165c042, 0x250e6ef4,
0x5957ee20, 0xda484fed3, 0xa666c502, 0x7e54e8ae,
0xd12ee9d9, 0xc0f1f38d4, 0x49829b5d, 0x1b5cdf3c,
0xd12ee9d9, 0xdcda2e3963, 0x28f4429f, 0xc88432c35,
0x4af40325, 0x9fc0dd70, 0xd8973ded, 0x1a02dc5e,
0xcd175b42, 0x10f1012bf, 0x6694d78c, 0xaccaab26b,
0x4ec11b9a, 0x3f168146, 0xc0ea8ec5, 0xb38ac28f,
0x1fed5c0f, 0xaab4101c, 0xee2db082, 0x470929e1,
0xe71843de, 0x508299fc, 0xe72fbc4b, 0x2e39f15dd,
0x9fa803fa, 0x9546b2de, 0xc0ea8ec5, 0xb38ac28f,
0x24d607ef, 0x8f97ebab, 0xf37f859b, 0x1c1f2e2f,
0xc25b71da, 0xf75e2269a, 0x7e1e393cd1, 0xed565b36,
0xf8c9def2, 0x46c9fc5f, 0x827b3a3, 0x70a56ddf,
0x0f25b510, 0x000f85a7, 0xb2e82e71, 0x68cb8816,
0x8f951e2a, 0x72f5f6af, 0xe4c2b3, 0xd34ff55d,
0x2e6b6214, 0x220b3e83, 0xd39ea6f5, 0x6fe041af,
0x6b2f1f17, 0xad3b99ee, 0x16a65ec0, 0x757016c6,
0xba770a4, 0xb0326e01, 0xf4b280d9, 0x4bfb1418,
<table>
<thead>
<tr>
<th>Hex Value 1</th>
<th>Hex Value 2</th>
<th>Hex Value 3</th>
<th>Hex Value 4</th>
<th>Hex Value 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 xd6aff27</td>
<td>0 xfd548203</td>
<td>0 xf56b9d96</td>
<td>0 x6717a8c0</td>
<td></td>
</tr>
<tr>
<td>0 x00d5bf6e</td>
<td>0 x10ee7888</td>
<td>0 xedf0fe64</td>
<td>0 x1ba93cd</td>
<td></td>
</tr>
<tr>
<td>0 x4b0d0184</td>
<td>0 x89ae930</td>
<td>0 x1c014f36</td>
<td>0 x82a87088</td>
<td></td>
</tr>
<tr>
<td>0 x5ead6c2a</td>
<td>0 xef22c678</td>
<td>0 x31204de7</td>
<td>0 x0c92e759</td>
<td></td>
</tr>
<tr>
<td>0 xd20028e</td>
<td>0 x303b446b</td>
<td>0 x0b00d9fc2</td>
<td>0 x9914a895</td>
<td></td>
</tr>
<tr>
<td>0 x906cc3a1</td>
<td>0 x54fefa70</td>
<td>0 x34c19155</td>
<td>0 x0e278a66</td>
<td></td>
</tr>
<tr>
<td>0 x131b5e69</td>
<td>0 xc3a8623e</td>
<td>0 x27bdfa35</td>
<td>0 x97f068cc</td>
<td></td>
</tr>
<tr>
<td>0 xca3a6ac</td>
<td>0 x4b55e936</td>
<td>0 x86602db9</td>
<td>0 x51df13c1</td>
<td></td>
</tr>
<tr>
<td>0 x390bb16d</td>
<td>0 x5a80b83c</td>
<td>0 x22b23763</td>
<td>0 x39d8a911</td>
<td></td>
</tr>
<tr>
<td>0 x2cb6bc13</td>
<td>0 xbf5579d7</td>
<td>0 x6c52cfa8</td>
<td>0 x0a841f96</td>
<td></td>
</tr>
<tr>
<td>0 xcbdb5476</td>
<td>0 x6864866</td>
<td>0 x4161e6ad</td>
<td>0 x897f51b</td>
<td></td>
</tr>
<tr>
<td>0 x956fcb7c</td>
<td>0 xfc68a306</td>
<td>0 x216799d9</td>
<td>0 x171a9133</td>
<td></td>
</tr>
<tr>
<td>0 x6c2466ed</td>
<td>0 x75eb5dcd</td>
<td>0 xdf1185f0</td>
<td>0 xe4f2b26</td>
<td></td>
</tr>
<tr>
<td>0 x26b9eef3</td>
<td>0 xadb36189</td>
<td>0 x8a7a9b</td>
<td>0 xe2c7308</td>
<td></td>
</tr>
<tr>
<td>0 xf77ded5c</td>
<td>0 x8b8bc58f</td>
<td>0 x06d6e421</td>
<td>0 xb4e1f9b</td>
<td></td>
</tr>
<tr>
<td>0 xb1cc715e</td>
<td>0 x68c0f999</td>
<td>0 x5d122f0f</td>
<td>0 xa4d2518</td>
<td></td>
</tr>
<tr>
<td>0 x097a5e6c</td>
<td>0 x0cbf18bc</td>
<td>0 xc2d7cfe0</td>
<td>0 x8bb7e42b</td>
<td></td>
</tr>
<tr>
<td>0 xa1ff23f</td>
<td>0 x35d9b8a2</td>
<td>0 x03da1a6b</td>
<td>0 x0688c02</td>
<td></td>
</tr>
<tr>
<td>0 x7dd1e354</td>
<td>0 x68a7d79a</td>
<td>0 x32c7753</td>
<td>0 xe62d655</td>
<td></td>
</tr>
<tr>
<td>0xa9829da1</td>
<td>0 x301590a7</td>
<td>0 x9bc1c149</td>
<td>0 x1350f1c</td>
<td></td>
</tr>
<tr>
<td>0 xd3779b69</td>
<td>0 x2d71f2b7</td>
<td>0 x183c58fa</td>
<td>0 xaacc4418</td>
<td></td>
</tr>
<tr>
<td>0 x8d8c8c76</td>
<td>0 x262039f0</td>
<td>0 x71a80d4d</td>
<td>0 x7a74c73</td>
<td></td>
</tr>
<tr>
<td>0 x449410e9</td>
<td>0 x20a4211</td>
<td>0 xf9c8082b</td>
<td>0 x0a6b334a</td>
<td></td>
</tr>
<tr>
<td>0 xb5f68ed2</td>
<td>0 x8243cc1b</td>
<td>0 x453c0ff3</td>
<td>0 x9be564a0</td>
<td></td>
</tr>
<tr>
<td>0 x4ff55a4f</td>
<td>0 x8740f8e7</td>
<td>0 xcca7f15f</td>
<td>0 xe30f2e21</td>
<td></td>
</tr>
<tr>
<td>0 x786d376d</td>
<td>0 xdf506f1</td>
<td>0 x8ee0973</td>
<td>0 x17be3c6</td>
<td></td>
</tr>
<tr>
<td>0 x7a670fa8</td>
<td>0 x5c31ab9e</td>
<td>0 xd4d6b18</td>
<td>0 xcc1f52f5</td>
<td></td>
</tr>
<tr>
<td>0 xe358eb4f</td>
<td>0 x19b9e343</td>
<td>0 x3a8d77dd</td>
<td>0 xcd83da6</td>
<td></td>
</tr>
<tr>
<td>0 x140fd52d</td>
<td>0 x395412f8</td>
<td>0 x2ba63360</td>
<td>0 x37e53a0</td>
<td></td>
</tr>
<tr>
<td>0 x80700f1c</td>
<td>0 x7624ed0b</td>
<td>0 x703dc1ec</td>
<td>0 x736795</td>
<td></td>
</tr>
<tr>
<td>0 xd659dd15</td>
<td>0 x66ce46d7</td>
<td>0 xd1abe76</td>
<td>0 xa44e0a0</td>
<td></td>
</tr>
<tr>
<td>0 x28f07c02</td>
<td>0 xc31249b7</td>
<td>0 x6e9ed6ba</td>
<td>0 xeaa47f8</td>
<td></td>
</tr>
<tr>
<td>0 xbbc7fbbd</td>
<td>0 xc507ca84</td>
<td>0 x965f4da</td>
<td>0 x8e8f35a</td>
<td></td>
</tr>
<tr>
<td>0 x6ad2aa44</td>
<td>0 x577452ac</td>
<td>0 xb5d674a7</td>
<td>0 x5461a46a</td>
<td></td>
</tr>
<tr>
<td>0 x6763152a</td>
<td>0 x9c12b7aa</td>
<td>0 x12615927</td>
<td>0 x7b4fb118</td>
<td></td>
</tr>
</tbody>
</table>
Summary:
All 32 functions have nonlinearity 112.
Each pair has hamming distance at most 10 from balance.
The first 8 functions have deviation from Cl(1) at most 12.
Boolean Function 0 has truth table:

```
0010011011000101011011001011011001100111110011000001010110010010100111011100100001101101110110110011110100101110011110001001110011000110010110111010011000101101001011101011100100111000110110111011010101101010111110010010010001011111011101001100001001001101110101110
```

$WHT_{max} = 32 \quad AC_{max} = 56$

Boolean Function 1 has truth table:

```
010011011110110010101101001100110011000010001100111110011000001110110010110110010010100111011100100001101101110110110011110100101110011110001001110011000110010110111010011000101101001011101011100100111000110110111011010101101010111110010010010001011111011101001100001001001101110101110
```

$WHT_{max} = 32 \quad AC_{max} = 48$

Boolean Function 2 has truth table:

```
00010011010101000000111101011101011101100101010010100110011001101110110110011110100101110011110001001110011000110010110111010011000101101001011101011100100111000110110111011010101101010111110010010010001011111011101001100001001001101110101110
```

$WHT_{max} = 32 \quad AC_{max} = 48$
Boolean Function 3 has truth table:

```
101011101110001100111001011001010010100101101001001011000110101
001111011101000101010101111111010010011101010001000110101110000
100110100000100101100111101001010001100110101110110100111
10100001000011101110111010101001101111011101101000110110
```

Boolean Function 4 has truth table:

```
11001000101110111111111101110011001100111010000111011111101010110
110010010110001111011010001100011001101110100001110111111101000110
010001101101000111010011010010101011100101011010111110000
0000101111100001011110111001100011111111111011010000011101000010111
```

Boolean Function 5 has truth table:

```
11010111010010001101111110101101100111111101101010010001101101111
1011010111001100111100000111000011101011101111101111111110
101101000101110011001010101111011001000001111100101100101101001011
110100101011111101111111101000001100001111101110100010111
```

Boolean Function 6 has truth table:

```
11000111000101001110010110111111011100101101010010001110111111000
1101000111101000011110001110010001111110100011011111111100
101100010101100111001011100100001111010110010100100011101110110000
00100100111011100110011111111111011110110001010111
```

Boolean Function 7 has truth table:

```
101100001011000010011000101100110101101011101010001001001101101
110100010111001111000111001000101111111110010100100011101110011001
110001110001101111000011111111110110101001000001110110110000010111
0001001110010101110010001111000011101101010101011110110110
```
$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 56$

Boolean Function 8 has truth table:

```
100001000010101011000111110111100010011100010111100111100011
010001110001110101100100000110011001011110011100010111100011
011011010111011111010100011101010101010011001110011110110000
010011100110100000011011001111100101100011011010011110111111
```

$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 56$

Boolean Function 9 has truth table:

```
0111011011000000100110110111000101000011001100101000010001100111
000110010111011100000010000110001011100011011111111111101000001
1010111010110011011100110011111011001011000010111000010011110
10001011101101101111000011011111111111011011000011000101110111
```

$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 56$

Boolean Function 10 has truth table:

```
1010001000001101110000100101111001110110001000110110011001110
11111111010111101011001111000101100010110101100010011111111
10000000111000001100011000111111111110010111111100100011111
001010010001011100111000111101100011111111010111000100001111
```

$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 56$

Boolean Function 11 has truth table:

```
01001001010100000111000010011011110011000111110100000101011110
10101010010111100110110111101011001111101111111110111010010110
1000001010100000111100011100111010110110001100110010111101110
001010010001110000000111111001101001000111111111111111111111
```

$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 56$

Boolean Function 12 has truth table:

```
11010000110100000000010011100111011001011000100111010011001100
111000011010011110100011111111110000010000011111111111111100
1011110001110011101000011110001111111111111111111111111111111
1100000011010010000000011100111000000011110100000111111111111
```

$WHT_{\text{max}} = 32$ $AC_{\text{max}} = 64$
$WHT_{max} = 32 \quad AC_{max} = 56$

Boolean Function 13 has truth table:

01111111101110101100100101111010010111110001001111111011101010010
10110010011011100111010000011110100110111000011001110000111001001
1011001010111011011000011001011010101010010110101010101011111101
11000011111011000010001000100001100111000110010010110110101101001

$WHT_{max} = 32 \quad AC_{max} = 56$

Boolean Function 14 has truth table:

110110101101101100111100000000000000000000011111110001010101110101
11101111101001000100101010100001100111011001100111011101111011111
01010001110010011110000001011010011010100100110100101111001001001
01100110110110110110110011001110111111100001101100101100110000100

$WHT_{max} = 32 \quad AC_{max} = 48$

Boolean Function 15 has truth table:

001010000100000100011100010100110111001001100101010110000111010011101
110011001101101000011100101010101010001100110100101001100110111111
1110000001111110010101011001000001001110011111110110111111001110010
000000001111100101111001111111101111111110111111111111111111111

$WHT_{max} = 32 \quad AC_{max} = 56$

Boolean Function 16 has truth table:

001011111010111000010101000111011001101100101010111100110100110011101
10001011110110000101101010011011001010101011011101101010011111100100
1110001001101111101010101010000011110101101110111111111111111111111
01101011001101101011101110100111010110011101111111111111111111111111

$WHT_{max} = 32 \quad AC_{max} = 48$

Boolean Function 17 has truth table:

011111111010100110110000110011000000001000101111010000111101111010101
00100100011111111011111100111111111111111111111111111111111111111111
01111101110001111111111111111111111111111111111111111111111111111111
01011111111100011111111111111111111111111111111111111111111111111111

$WHT_{max} = 32 \quad AC_{max} = 48$
$WHT_{max} = 32$ $AC_{max} = 64$

Boolean Function 18 has truth table:

```
0001110111001010001000010000001100100111010110111110100101011010
000001001001111000011001100100110100011111110001001100100110101
1010111000010100110000010100000000100010111100001101110101101
001110011011101101011100011110000111110101100110111100001
```

$WHT_{max} = 32$ $AC_{max} = 80$

Boolean Function 19 has truth table:

```
10000100011110100011111110101000110001001110110000101001101001000
11110001101110001000001110010101000111011011010001101010111
00011011100111100111111101001010010001111110100101111110000
101010111011101101111010011000100010000111100010111010010101101
```

$WHT_{max} = 32$ $AC_{max} = 56$

Boolean Function 20 has truth table:

```
10011101111010111000101010010111000001000000001101000101010100001100
111011000111110011010100011001110111000101110110110010011011010
10001101110011110100010101101010110001011101101101011100110011100
10101101110010000100100101110011001001011100110011010010011111
```

$WHT_{max} = 32$ $AC_{max} = 56$

Boolean Function 21 has truth table:

```
00011100000100001000110010001010000000001100101100101000111100001
1011010100110001100100110001001100111001101010000100000010001
1111011101100011111001001100011100000111100110001101000111110001
1111011101111000100100000110001001110000111100010001110000001
```

$WHT_{max} = 32$ $AC_{max} = 64$

Boolean Function 22 has truth table:

```
010111000001110101000111101100001000111111101011110100010111001110
1011010011001011100101001001011111010010000111011010001011110000
00010110101010101100011100000000110110010111010110111111111110001
01100111110011000100000101011000001010011100110011100101100
```
WHT_{max} = 32 \quad AC_{max} = 40

Boolean Function 23 has truth table:

01000010101101001110101011000011101111110101111011000110101
11001000011010111000110101111000111101110001101111011101101
1101111100000000111010111000110000011010001010101100111101111
1000011010000011111000000011111011000000110001101011001

WHT_{max} = 32 \quad AC_{max} = 56

Boolean Function 24 has truth table:

1001010111000010011010110011100011111111111010111101000110111
011101000110110010101111000011000011001010010011001101111111
010101010111000001100011001011100000011110000011101001100111
0101011111011001001101111101110010110011001101001101111111

WHT_{max} = 32 \quad AC_{max} = 48

Boolean Function 25 has truth table:

0111010011111000111111111111111111111111010011110001010110111
011001000110110010101111000011000011001010010011001101111111
1001100010100011111001101101111111011011001100001111101111011
0110001011110011010000101100101100101110011001101111111111111

WHT_{max} = 32 \quad AC_{max} = 48

Boolean Function 26 has truth table:

000110010001101111001101000000010100010001100001010011001011111
010010001100101011001111000100110110001111111110010000111011
0011000011111111111000110101100111001100110100101110011111111
00110111100001111110011111110110011111111010000111111111111

WHT_{max} = 32 \quad AC_{max} = 64

Boolean Function 27 has truth table:

010101001111111111010100100010001111111111111111111111111111111111
1000001100100000110101100110011011000110011111111111111111111111111
1001110001010100111111111111111111111111111111111111111111111111111
1110011111101000001011100110011011101000111111111111111111111111111
$WHT_{\text{max}} = 32 \quad AC_{\text{max}} = 48$

Boolean Function 28 has truth table:

```
010000011011101000110111100000001101110010100101010101010101100100
010000110000110100011101110010001101110101101011010111100000000
110111011001010001110011001001100110011110011010101010110101110
0000001111111111111111101010111110110001101100011101110010111110
```

$WHT_{\text{max}} = 32 \quad AC_{\text{max}} = 56$

Boolean Function 29 has truth table:

```
1101100101110010001101111000101010101010101010101010101010101011
110110010101011111010001001101111111000110111110011010100000000
1100101111110001001010101001010101000000000111101001011101110
111001011011100010010101101000110001100101010111111100100101001
```

$WHT_{\text{max}} = 32 \quad AC_{\text{max}} = 56$

Boolean Function 30 has truth table:

```
101011100000110000111011010101010101010101010101010100000000000
1101000111010001010111000010000000101010111001000010100100000010
01010100000111101000010101011111110011001101101001101000000000
011011100011011111110110100101110100111111111101010101010100100
```

$WHT_{\text{max}} = 32 \quad AC_{\text{max}} = 48$

Boolean Function 31 has truth table:

```
10000100101111110100110010101110011001111100111001100110000000110
100001110100000011010110101010010101011100011111001110011100000110
0111010000010100111111110111000111011101101100011101000101010001
110000010000011001100111010100100110001100011101001001010011111
```

$WHT_{\text{max}} = 32 \quad AC_{\text{max}} = 48$
Distributions for total component boolean functions:

<table>
<thead>
<tr>
<th>value</th>
<th>WHT</th>
<th>AC</th>
<th>ANF imbalance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1169</td>
<td>1420</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>682</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2068</td>
<td>2569</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>590</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1504</td>
<td>1937</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>426</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>956</td>
<td>1218</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>222</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>575</td>
<td>632</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>247</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>256</td>
<td>0</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>value</td>
<td>$WHT_{max}$</td>
<td>$CI(1)_{dev}$</td>
<td>$AC_{max}$</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Appendix B

Example MARS-Like S-Box, SB

$SB[i]$ (i = 0..511):

0x657ce571 0xb2c0a31b 0xea0ad0a0 0xd4d49175 0x4794396c 0xad63322
0xb6467628 0x58b1bdcb 0x3216bd0c 0x87810f0e 0x8928aab6 0x309926d6
0x86ed7cda 0x7ce28025 0xab91f5e5 0x0a559d1f 0x03b7fccc 0xc635a7c2
0xb12e7967 0xf3c464ce 0x1c8815d1 0x12fa97fb 0x6937c3b8 0x8df7406d
0x581a310f 0x60add4 0x61ecf4be 0x4abcb39d 0x3fbf5af2
0xb01c48c9 0x01193559 0x6a0de825 0x12fa97fb 0x6937c3b8 0x8df7406d
0xd1557eb8 0x6d1f101c 0xee7fd7ab 0x3ac220a1 0x03e23430 0xd6746be1
0x5e026256 0x5798f80 0xe053ea7f 0xae0d8259 0x192bb9e3
0xc28a5f35 0x54c8ceef 0x00ee5da73 0x4d315a7e 0x8eda0aba3 0xe5e18c07
0x9e923d96 0xf94dc633 0xb02e60bc 0x1b6acf89 0xb8c718a2 0xad77b720
0x2444c1d0 0x9d64bd69 0xc6b3f1be 0x85fbf907 0x6a2ab1a0
0x105349ff 0x0c07d708 0xb9af64fd 0x81f3c534 0x1a450da2 0xdf5d20e38
0xe8a00d0 0x149dc90a 0x2b69526e 0x9d7d7598 0xefe968d7 0x7f55539a
0x819c2b62 0x7f85c4ff 0x6c8d1bb8 0xc57b529d 0x664294e4 0x7eb2d2cc
0x6f7ade7 0x4eeb926 0x8858f38d 0x2c47b42 0xbeb0b0e6 0x75003971
0x1eb2e50 0x02e6f63d 0x05d868ce 0x4d0ccac2 0x502f81f8 0x25724c59
0x9852165f 0xa9bd3bb2 0x40308156 0x3a19ebb9 0x3bb137f0 0x8171f78f
0x751ed38a 0xe74ac36 0x59745744 0xda8f3b85 0xf4771cfe 0x6510184d
0xc36d332b 0xbfb86d81 0xe95e9ec7 0x0c032dec 0x8c24e5f4 0x6a746cbb
0xe9a5b509 0x0fbc5c93 0x8b138d45 0x8f6a906e 0xed378e6b 0x131faa01
0xe79f8558 0x64b15239 0x255e0943 0x7be2d50a 0x6d28a6bc 0xba5349d
Appendix B. Example MARS-Like S-Box, SB

0x8cc7e39a 0xda90c2e8 0xda009eb7 0xc03e9b1d 0x0b79da15 0xda0130e7
0xcb03e5b7 0x9ebf9748 0xb9f897e8 0x760754b8 0x0d9c62b8 0x64d36fe8
0x99b6f97e 0x4f8605e8 0x79f6f97e 0x127e62f8 0x699e62f8 0x699e62f8
0xf69e62f8 0x127e62f8 0x699e62f8 0x79f6f97e 0x99b6f97e 0x4f8605e8
0x760754b8 0xda0130e7 0x0d9c62b8 0xc03e9b1d 0xda009eb7 0x8cc7e39a
Bibliography


